ESCUELA POLITÉCNICA SUPERIOR

UNIVERSIDAD CARLOS III DE MADRID

PROYECTO FIN DE CARRERA

GESTIÓN DE INFORMACIÓN ACTIVA EN BASES DE DATOS POST-RELACIONALES: CACHÉ 2007

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Autor: Gema Raboso Domínguez

Tutor: Manuel Velasco de Diego Enero 2009

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

En esta introducción se hace referencia a conceptos generales de las bases de datos, para posteriormente introducir el modelo de Bases de Datos Activas.

1.1. Problema general

Toda organización, sea pequeña o grande, tiene unas necesidades de información, bien en la forma tradicional de datos administrativos, bien en sistemas avanzados de tratamiento de información de todo tipo. De todos los datos que entran y salen de esa organización, en el formato que sea, unos son importantes y otros no tanto.

El objetivo de un analista es identificar la información importante y estructurarla de forma que sea útil para todos los miembros de la organización. Ese sistema de información puede ser mecanizado mediante herramientas informáticas y servir así a la productividad de la entidad.

En un principio, los sistemas de información a mecanizar eran sencillos y reflejaban más o menos exactamente el flujo administrativo de papel del exterior hacia la empresa, dentro de la misma empresa, y de la empresa hacia el exterior nuevamente. Para ello se utilizaban los lenguajes de programación disponibles, más o menos adecuados para la tarea, que manejaban ficheros organizados según lo permitía la tecnología del momento.

Pero pronto nuevas necesidades y expectativas hicieron que el mantenimiento y creación de aplicaciones informáticas, junto con el incremento masivo de la cantidad de datos a almacenar y tratar, se convirtiera en un cuello de botella debido a problemas de redundancia (e inconsistencia) de datos, deficientes medidas de seguridad, baja calidad de la información almacenada, y pérdidas de información por diversas causas. La tecnología del momento no era adecuada para sistemas de información en constante evolución y con unos requerimientos de rendimiento y fiabilidad cada vez más exigentes.

La aparición de las técnicas de bases de datos vino a solucionar gran parte de estos problemas.

Una **base de datos** es una colección de información organizada de forma que se pueden seleccionar rápidamente los fragmentos de datos que se necesitan.

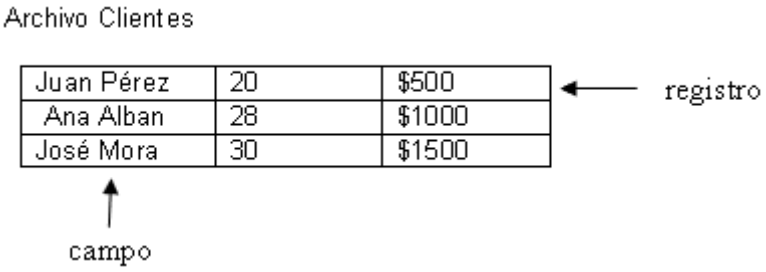
El sistema de base de datos comprende cuatro componentes principales: datos, hardware, software y usuarios.

Los **datos** son del tipo:

Campo: unidad mínima de referencia.

Registro: conjunto de campos.

Archivo: conjunto de registros.



El **hardware** está constituido por dispositivos de almacenamiento como son los discos.

El **software** es el DBMS ó Sistema Administrador de Bases de Datos.

Los **usuarios** son las personas o aplicaciones que interactúan con el sistema y pueden realizar diferentes operaciones tales como:

- Agregar nuevos archivos vacíos a la base de datos, es decir insertar nuevas tablas.
- Insertar datos dentro de las tablas existentes.
- Recuperar datos de las diferentes tablas existentes.
- Modificar datos en tablas existentes.
- Eliminar tablas existentes.

La información que se almacena puede ser cualquier cosa que se considere de importancia para el individuo u organización.

Los primeros modelos de datos que surgieron fueron los **modelos jerárquicos**. Aparecieron a mediados de los 60, y como su nombre indica, todas las interrelaciones entre los datos se pueden estructurar según jerarquías, según estructura de datos de árbol. Un registro hijo depende de un registro padre (un hijo sólo puede tener un padre pero un padre puede tener varios hijos).
Para acceder a la base de datos tendríamos que localizar el elemento que a mi me interesa e ir descendiendo por el árbol a través de los hijos del nodo padre; después de ver todo lo del hijo 1 vuelvo al padre y sigo por el hijo 2 (y así sucesivamente). El camino de búsqueda es largo pues tengo que navegar por todos los registros.
Los problemas de una Base de Datos Jerárquica eran que sólo podían tener relaciones de 1 a muchos, pues si intentábamos hacer relaciones de muchos a muchos había que duplicar la información, y esto podía llevar a problemas de integridad y de consistencia.

Los siguientes modelos de datos fueron los **modelos en red (grafos),** que permitían que un registro estuviera subordinado a otra serie de registros de más de un archivo (relación de muchos a muchos). Se relacionaban los registros a través de punteros, y para saber quien era el padre o el hijo se marcaban direcciones.
En 1971 se le dio una forma más o menos estándar, a la que se llamó Modelo CODASY.
Ventajas que aportó este modelo:

- flexibilidad, permitía representar más relaciones que el modelo jerárquico.
- Estaba normalizado.
- Rendimiento prácticamente similar al de las Bases de Datos Jerárquicas.

Desventajas:

- Modelo rígido, una vez definida la estructura de las relaciones ya no se podían hacer modificaciones, había que reestructurarla; las Bases de Datos Relacionales le dieron la solución a este problema.

Edgar Frank Codd, trabajador de IBM, definió las bases del **modelo relacional** en los años 70 y rápidamente causó grandes impactos debido a su simplicidad. Pocos años después, el modelo se empezó a implementar hasta ser el modelo de bases de datos más popular.
Este modelo utiliza el concepto de relación matemática y tiene sus bases teóricas en la teoría de conjuntos y la lógica de predicados de primer orden.

En las bases de Codd se definían los objetivos de este modelo:

- Independencia física: la forma de almacenar los datos no debe influir en su manipulación lógica.
- Independencia lógica: las aplicaciones que utilizan la base de datos no deben ser modificadas por que se modifiquen elementos de la base de datos.
- Flexibilidad: la base de datos ofrece fácilmente distintas vistas en función de los usuarios y aplicaciones.
- Uniformidad: las estructuras lógicas siempre tienen una única forma conceptual (las tablas)
- Sencillez

Las bases de datos relacionales se basan en el uso de **tablas** (también se las llama relaciones). Las tablas se representan gráficamente como una estructura rectangular formada por filas y columnas. Cada columna almacena información sobre una propiedad determinada de la tabla (se le llama también **atributo**), nombre, dni, apellidos, edad,.... Cada fila posee una ocurrencia o ejemplar de la instancia o relación representada por la tabla (a las filas se las llama también **tuplas**).

El **grado** es le número de atributos de la tabla.

La **cardinalidad** es el número de tuplas de una tabla.

El **dominio** es el conjunto válido de valores representables por un atributo.

Las **restricciones** son unas condiciones de obligado cumplimiento por los datos de la base de datos. Las hay de varios tipos.

- **Inherentes**: son aquellas que no son determinadas por los usuarios, sino que son definidas por el hecho de que la base de datos sea relacional.
  - No puede haber dos tuplas iguales
  - El orden de las tuplas no importa
  - El orden de los atributos no importa
  - Cada atributo sólo puede tomar un valor en el dominio en el que está

- **Semánticas:** permiten a los usuarios incorporar restricciones personales a los datos:

- Clave primaria: hace que los atributos marcados como clave primaria no puedan repetir valores.

- Unicidad: impide que los valores de los atributos marcados de esa forma puedan repetirse.

- Obligatoriedad: prohíbe que el atributo marcado de esta forma no tenga ningún valor.

- Integridad referencial: prohíbe colocar valores en una clave externa que no estén reflejados en la tabla donde ese atributo es clave primaria.

- Regla de validación: condición que debe de cumplir un dato concreto para que sea actualizado.

1.2. Problema específico

Una vez introducidas las bases de datos y el modelo de base de datos relacional se explicará de forma general en que consiste el comportamiento activo de las bases de datos.

El primer estándar de SQL (lenguaje de definición, manipulación y control de datos) en el que se consideró la visión de las bases de datos activas fue en SQL:99 ó SQL 3.

Las bases de datos convencionales se consideran pasivas, ya que su evolución se programa en las aplicaciones; por el contrario, en los sistemas de bases de datos activas, esta evolución se define en el propio esquema de la base datos, dado que los sistemas gestores de bases de datos contienen las herramientas necesarias para poder definir el mencionado comportamiento activo.

Éste comportamiento permite un amplio número de posibilidades, ya que si se hace un buen diseño del comportamiento activo de la base de datos, se consiguen grandes ventajas como:

- Un mejor mantenimiento y reutilización de código, puesto que el conocimiento se traslada a la base de datos y esto permite hacer que no sea necesario tener duplicada funcionalidad en las diferentes aplicaciones que interactúen con ella.
- La integridad de la base de datos es preservada por el propio Sistema Gestor de Base de Datos, con independencia del medio de acceso.
- Permiten simplificar las aplicaciones, eliminando de ellas funcionalidades fundamentadas en la propia semántica de los datos.

Por tanto, se puede definir una base de datos activa, como aquella que es capaz de detectar situaciones de interés y actuar en consecuencia. Para poder lograr este funcionamiento alguien debe determinar cuáles son las situaciones a detectar y cuál es el comportamiento a seguir en cada una de ellas. El mecanismo que se utiliza para diseñar y especificar el comportamiento activo son reglas del tipo Evento – Condición – Acción.

2. ESTADO DE LA CUESTIÓN

En este apartado se exponen los fundamentes teóricos a partir de los cuales se ha llevado a cabo el desarrollo del proyecto. Se hará una primera introducción a las bases de datos, posteriormente se hará una breve introducción a las bases de datos relacionales y se terminará con el tema central del proyecto las bases de datos activas.

2.1. Introducción a los sistemas de bases de datos

Uno de los usos más frecuentes de los ordenadores es el de almacenamiento masivo de datos. Aquí es importante hacer distinción entre datos e información. Cuando nos referimos a datos, estamos hablando de un texto, un número o un conjunto arbitrariamente grande de ambos.

En el momento en que hablamos de información, nos referimos a datos que pueden ser relacionados entre sí, de forma más o menos inteligente, con el objetivo de sacar alguna conclusión que nos permita tomar decisiones importantes en nuestra empresa u organización.

Normalmente, los datos necesarios no se almacenan en un sólo fichero, sino que para extraer una determinada información, es necesario consultar ficheros de distinto tipo.

Para poder realizar todas estas operaciones es necesario almacenar de alguna forma datos sobre los clientes, los paquetes de viajes, las reservas, información estadística, etc. según la información que se desee obtener.

Tradicionalmente, antes del uso de ordenadores, la información se extraía tras un minucioso examen de los datos almacenados en fichas de cartón que se guardaban en los cajones de grandes ficheros en una oficina, lo cual era muy costoso debido al tiempo requerido, o bien a la poca exactitud de los resultados obtenidos, que hacía tomar decisiones poco acertadas.

Con la llegada de los ordenadores, y los dispositivos de almacenamiento electrónico, discos duros, tambores magnéticos, fichas perforadas, cintas magnéticas, etc., aparece el concepto de fichero electrónico. Un fichero electrónico es igual que un fichero con cajones y fichas de cartón en su interior, sólo que están almacenados sobre una superficie magnética (con el consiguiente ahorro de espacio), y son gestionados a través de un ordenador (con el consiguiente ahorro en velocidad a la hora de acceder), lo que posibilita su examen de forma veloz a través de la máquina.

Normalmente, los datos necesarios no se almacenan en un sólo fichero, sino que para extraer una determinada información, es necesario consultar ficheros de distinto tipo. Este fue un gran paso en lo que respecta al almacenamiento y gestión de los datos.

No obstante, a medida que pasó el tiempo, se observó que la información extraída de estos ficheros era a menudo contradictoria y propensa a errores de coordinación (incoherentes). Para solucionar este problema, interviene de nuevo la capacidad de los ordenadores para hacer rápidamente muchas cosas simples. Se optó por hacer que el ordenador controlase la concordancia entre los datos, y para ello se dio a los datos una estructura distinta, ya no basada en ficheros como tales, sino basada en listas o tablas de fichas (llamadas tuplas o líneas), y en cada una de las cuales no se encontraba toda la información, sino que era necesario relacionar datos de varias tablas para obtener la información deseada. Aparecen así las bases de datos.

La evolución de la informática ha pasado por numerosas formas de abordar los problemas mediante programas, en función de la potencia de las máquinas en cada época, y de la experiencia de los informáticos que ha ido aumentando a lo largo de los años. De esta forma, aparecieron las bases de datos jerárquicas y en red. Posteriormente las bases de datos relacionales, que son las que abordaremos, sustituyeron rápidamente a estas dos, ya que las relacionales se basan en estudios matemáticos que aseguran la eficiencia de las operaciones que permite realizar, así como su correctitud y completitud, y definen

claramente la forma en que deben abordarse los problemas. Actualmente, las bases de datos relacionales son las más usadas a nivel profesional, aunque para aplicaciones avanzadas y en los ambientes académicos también tienen bastante importancia las bases de datos orientadas a objetos (especialmente para trabajos sobre Internet), y las bases de datos deductivas (para trabajos en Inteligencia Artificial).

En resumen, un SGBD es un programa de ordenador que facilita una serie de herramientas para manejar bases de datos y obtener resultados (información) de ellas. Además de almacenar la información, se le pueden hacer preguntas sobre esos datos, obtener listados impresos, generar pequeños programas de mantenimiento de la BD, o ser utilizado como servidor de datos para programas más complejos realizados en cualquier lenguaje de programación.
Además, ofrece otras herramientas más propias de la gestión de BD como sistemas de permisos para autorización de accesos, volcados de seguridad, transferencia de ficheros, recuperación de información dañada, indización, etc

2.1.1 Propiedades de las Bases de Datos.

Los objetivos que debe cubrir una base de datos son:

- ***Evitar la redundancia e inconsistencia en los datos***.

No sólo se debe aprovechar la enorme capacidad de los ordenadores para almacenar grandes cúmulos de información. Éstos pueden emplearse en controlar que los datos existentes sean consistentes entre sí, y que no haya información contradictoria entre los datos almacenados.

Por otro lado, es el usuario quien tiene que decidir qué se almacena en la base de datos y qué no. Evidentemente sólo deben almacenarse aquellos datos que se necesitan para algo, aquello que nos será útil en nuestro trabajo.

El usuario es quien debe decidir además en que forma se almacenan los datos.

Evitar redundancias (datos o información repetida) en los datos almacenados es un objetivo prioritario ya que con ello se consigue:

a) No desperdiciar capacidad de almacenamiento (cinta, disco, etc.).
b) Evitar la extracción de una misma información por dos caminos distintos, con lo que la información extraída puede ser contradictoria.

- ***Facilitar el acceso a los datos.***

En los sistemas antiguos, los datos almacenados en ficheros electrónicos sólo podían ser accedidos a través de programas de ordenador que realizaban alguna tarea específica muy concreta. Para cada tipo de información que se necesitaba, era necesario crear un programa que la extrajera de los datos de los ficheros.

Evidentemente, esta forma de trabajo es lenta, y tiene un cuello de botella en el centro de proceso de datos, con el agravante de que da lugar a una miríada de programas distintos, cada uno para atender una necesidad particular. Además, si por cualquier causa, cambia la estructura de algunos de los programas ya hechos, no servirán porque fueron creados en base a una estructura determinada, que ahora se ha visto alterada.

Para evitar todos estos problemas, se inserta entre el nivel a que se hallan los datos y las aplicaciones, un programa auxiliar de máxima importancia: es el Sistema Gestor de Bases de Datos (S.G.B.D.). Una de sus funciones es la de suministrar un lenguaje que permita consultar fácilmente la base de datos sin necesidad de hacer un programa para cada consulta; bastará con formular la consulta según este lenguaje simple, para obtener un resultado más o menos inmediato fruto de las gestiones realizadas por el S.G.B.D. sobre los datos.

- ***Facilitar la creación de programas que trabajen sobre los datos.***

Como se ha indicado anteriormente, los datos pueden estar repartidos en múltiples tablas o en múltiples ficheros; incluso puede haber varias bases de datos de cuyo conjunto sea necesario extraer información. Es más, cada base de datos puede estar creada con un sistema.

De esta forma, es difícil crear aplicaciones que trabajen sobre los datos. De nuevo, esto se soluciona con un S.G.B.D. que aísla al creador de aplicaciones de este caos de datos, y le da una visión uniforme de los datos, facilitando de esta manera la programación de aplicaciones complejas.

- ***Permitir el acceso concurrente.***

Un ordenador es una máquina que sólo es capaz de hacer una cosa cada vez: leer de memoria, escribir en ella, realizar un cálculo, etc.; en definitiva cosas muy simples. Sin embargo, en la mayoría de los trabajos se requiere que varias personas puedan trabajar a la vez, quizás aparentemente no con la misma máquina, pero sí sobre los mismos datos.

Dado que varias personas trabajan a la vez, es posible que el ordenador central reciba peticiones de acceso sobre los datos de forma simultánea: es lo que se llama acceso concurrente.

Para observar los problemas que puede producir el acceso concurrente, supongamos que el ordenador central es un escribiente al que cada usuario le dicta la ficha que debe tomar y lo que debe hacer sobre ella.

Si no hubiese requerimientos de tiempo por parte de cada usuario, no habría problema, pues al escribiente le bastaría con atender las peticiones por riguroso orden de llegada. Sin embargo, para dar la sensación de que el escribiente atiende a todo el mundo por igual, este hace uso de una técnica llamada tiempo compartido, o sea, reparte cada minuto de su tiempo en partes iguales, una para cada usuario que le ha hecho una petición. De esta forma, pueden aparecer problemas en los accesos concurrentes a los datos.

Este problema se ve agravado enormemente en el momento en que se introduce el concepto de transacción. Una transacción es un conjunto de operaciones sobre los datos que debe ser tratada como una única operación.

Todo esto debe ser gestionado también por el S.G.B.D., que debe asegurar que el resultado final debe ser consistente, y debe informar al usuario afectado si se ha producido algún error en la transacción que intentó realizar.

- ***Controlar la seguridad en los accesos a los datos.***

Cuando se trabaja con ficheros electrónicos, el ordenador que los almacena, como se indicó en el apartado anterior, suele recibir peticiones de otros terminales, con lo que el acceso a los datos no está restringido sólo a los que tocan el ordenador directamente, pudiendo ser utilizado desde un terminal remoto situado, posiblemente, en cualquier parte del mundo. Por tanto los métodos de seguridad tradicionales no son suficientes, y es necesario recurrir a medios electrónicos de protección que impidan acceder a los datos más importantes, excepto a aquellas personas que estén autorizadas.

Para conseguir esto, se asignan una serie de prioridades a cada usuario, así como un nivel de seguridad a cada tipo de dato, de manera que cada usuario sólo puede acceder a los datos de nivel inferior o igual a su prioridad. Además, existe un registro electrónico que toma nota de los accesos a los datos de mayor nivel, dejando así constancia de la fecha, hora y usuario que efectuó el acceso, o intentó un acceso frustrado.

Todo esto es llevado a cabo por el S.G.B.D. y debe ser inspeccionado regularmente por el departamento de proceso de datos, en busca de intentos de violación del sistema.

- ***Asegurar la integridad de la base de datos.***

Los datos deben poseer unas características determinadas en función de lo que se pretende hacer con ellos, y del área de trabajo en que se desarrolla la empresa. Nos referimos a características que hacen que los datos almacenados representen una información coherente para el usuario que los maneja. Es lo que se llama la Integridad de la base de datos. Por ejemplo se debe impedir que en el NIF de un cliente se coloque su Nombre. Todas estas son restricciones de integridad que el S.G.B.D. debe mantener en la base de datos.

2.1.2. Definición de Base de Datos

Existen numerosas definiciones de lo que es una base de datos, y de lo que es un S.G.B.D. De hecho, a menudo se confunden ambos términos dada la relación biunívoca existente entre ambos.

Nosotros consideraremos como base de datos una colección de datos junto con el S.G.B.D. Idealmente, el S.G.B.D. debe poseer una serie de características indispensables para satisfacer a los usuarios:

- Debe poseer un lenguaje de definición de datos que permita fácilmente la creación de nuevas bases de datos, así como la modificación de su estructura.

- Debe poseer un lenguaje de manipulación de datos, que permita la inserción, eliminación, modificación y consulta de los datos de la base, de la forma más eficiente y conveniente posible.

- Debe permitir el almacenamiento de enormes cantidades de datos (miles de millones de caracteres), sin que el usuario perciba una degradación en cuanto al rendimiento global del sistema.

- Debe permitir la gestión segura de los datos, con respecto a accesos no autorizados, y a accidentes producidos por los dispositivos mecánicos o electrónicos que soportan los datos almacenados.

- Debe permitir el acceso simultáneo por parte de varios usuarios, impidiendo además, que dichos accesos concurrentes den lugar a datos corruptos o inconsistentes.

- Debe suministrar independencia física de los datos, que asegure que, sea cual sea la estructura de los datos en los dispositivos electromecánicos de almacenamiento, el usuario y las aplicaciones los percibirán siempre de manera uniforme y útil.

La forma en que las distintas bases de datos abordan estas características difiere enormemente, no sólo por las técnicas utilizadas sino también por las aproximaciones o paradigmas con que se han desarrollado.

Nos centraremos exclusivamente en el tipo más extendido: las bases de datos relacionales, ya que tienen un formalismo subyacente que las hace muy potentes.

De hecho, todas las características que hemos visto que debe poseer un S.G.B.D., son suministradas a través de entornos e interfaces amigables y comprensibles que permiten un rápido aprendizaje de todas las funciones propias de una base de datos.

2.1.3. Componentes de un Sistema de Base de Datos

A continuación se expondrán los principales componentes de un sistema de base de datos.

- ***Datos.***

Efectivamente, una base de datos no tiene sentido sino está compuesta por datos. Lo que no está tan claro es la forma en que estos datos se deben disponer, qué datos se deben almacenar, y cómo los debe entender la máquina.

La disposición de los datos depende del ámbito de aplicación concreto en que se enmarque la base de datos. No es lo mismo una base de datos que almacene un dibujo vectorial de monumentos históricos que una base de datos para almacenar las reservas de clientes en un hotel. Lo que varía

fundamentalmente en uno y otro caso, es la forma en que los datos se relacionan entre sí, y el tipo de accesos a la información que va a realizar el usuario. Además, los datos deben disponerse de manera que las consultas sean lo más eficiente posible, evitando a la vez la existencia de datos duplicados que pueden dar al traste con la coherencia de la base de datos. De esta forma, no sólo se consideran datos aquellos que el usuario desea almacenar, sino toda estructura de apoyo que el sistema necesite para hacer más eficiente una consulta.

Por otro lado, es muy importante decidir qué datos son los que se van a almacenar. A menudo ocurre que en una base de datos se almacenan las facturas con una antigüedad inferior a X años. El número de años que se desea almacenar una factura electrónicamente puede ser una decisión crucial, dependiendo de la capacidad de almacenamiento de nuestra máquina y del volumen de facturas que se expidan anualmente. Además, si casi no se necesita acceder a dicha información histórica, puede ser más interesante destruir electrónicamente esa información, y almacenarla exclusivamente en papel, con lo que el ahorro de espacio en disco puede contrarrestar el esfuerzo de hacer las consultas manualmente.

Por último, también es interesante, por cuestiones de seguridad y aprovechamiento de los recursos del ordenador, decidir en qué formato se van a almacenar los datos: si se van a almacenar encriptados para que nadie los pueda copiar, o si se van a codificar o a comprimir de alguna forma para hacer que ocupen menos espacio. Existe toda una teoría sobre compresión y codificación de datos, que puede hacer que, dependiendo de las características de los datos a almacenar, se requiera mucha menos memoria de la necesaria siguiendo los métodos convencionales de almacenamiento.

- ***El sistema gestor de bases de datos.***

El sistema gestor de bases de datos (S.G.B.D.) ha sido ya introducido, y su importancia destacada en todas las características que debe poseer una base de datos.

- Interactuar con el Sistema Operativo.

El Sistema Operativo es el programa principal que se encarga de controlar que el ordenador funciona bien. Así, para asegurar que no pasan cosas raras, el único que puede tocar estos dispositivos en el S.O. es el S.G.B.D.

- Mantener la seguridad.

Evitar accesos fraudulentos a los datos, así como la extracción de información codificada.

- Permitir las copias de seguridad.

Se trata de efectuar una copia de los datos a un dispositivo auxiliar de almacenamiento, pensado precisamente para guardar fiel copia del contenido de la base de datos en un momento determinado.

- Controlar la concurrencia.

Debe permitirse el acceso simultáneo a los datos por parte de varios usuarios, lo que conlleva numerosos problemas de coherencia y coordinación.

-Lenguajes de bases de datos.

La interacción del usuario con la base de datos debe efectuarse a través de alguna técnica que haga fácil la comunicación, y que permita al usuario centrarse en el problema que desea solucionar, más que en la forma de expresarlo con las técnicas que se le suministran. La mejor forma de alcanzar este objetivo, es darle un lenguaje parecido al lenguaje natural, que le permita expresar de forma sencilla los requerimientos. En función de estos requerimientos, podemos tener, fundamentalmente dos tipos de lenguajes para comunicarnos con el S.G.B.D.

- Lenguaje de definición de datos (LDD).

Permite la construcción de frases o sentencias que le indican al S.G.B.D. las características particulares de la base de datos sobre la que se está trabajando, así como la creación de nuevas bases de datos. La creación de esquemas y su modificación, la creación y supresión de índices, la especificación de unidades de almacenamiento en los ficheros, así como la asignación y privación de prioridades se realizan a través de este lenguaje.

- Lenguaje de manipulación de datos (LMD).

El lenguaje de manipulación de datos es el que usan los usuarios directos para efectuar sus operaciones sobre la base de datos. Estas operaciones son básicamente de inserción, eliminación, modificación y consulta de datos, aunque también se pueden introducir capacidades para crear visiones de los datos que faciliten otros accesos.

2.2. Bases de Datos Relacionales

2.2.1. El Modelo de Datos

Una base de datos está definida generalmente por un conjunto de datos que representan mediante un modelo determinado un universo dado. Un modelo de datos no es más que un método conceptual para estructurar los datos. Existen tres modelos de datos fundamentales:

- Modelo jerárquico.

Consiste en que todas las interrelaciones de los datos se basan en jerarquías. Los archivos se conectan entre sí mediante punteros físicos (dirección física que indica donde puede encontrarse un registro sobre el disco) o campos añadidos a los registros individuales.

En una jerarquía un padre (registro propietario) puede tener muchos hijos (registro subordinado) pero un hijo sólo puede tener un padre. Este modelo

tenía algunas limitaciones ya que no todas las interrelaciones se pueden representar en una estructura jerárquica. Para intentar solucionar estas limitaciones se desarrollan los sistemas de base de datos en red.

- Modelo en red.

Los sistemas de base de datos en red al igual que los jerárquicos utilizan punteros físicos En este caso, un padre puede tener muchos hijos y un hijo puede tener muchos padres, es decir, un registro puede estar subordinado a registros de más de un archivo.

A principios de los 70 se desarrollaron y se comercializaron varios SGBD en red y este modelo de datos se normalizó como el modelo CODASYL. Ejemplos de bases de datos en red son ADABAS, TOTAL, IMAGE,...

- Modelo Relacional.

El uso de punteros era simultáneamente una fortaleza y una debilidad de los sistemas de bases de datos jerárquicos y en red. Los punteros permitían una rápida recuperación de los datos, pero las interrelaciones tenían que definirse antes de que el sistema se pusiera en explotación. Era muy difícil recuperar datos basados en otras interrelaciones.

En 1970, E.F. Codd publica un artículo en el que argumenta que los datos deberían relacionarse mediante interrelaciones naturales, lógicas e inherentes a los datos, más que mediante punteros físicos. Codd propone así, un modelo simple de datos en el que todos ellos se representarían en tablas constituidas por filas y columnas.

A estas tablas se les dio el nombre de relaciones y por eso se denominó al modelo relacional. Codd también propuso dos lenguajes para manipular los datos en las tablas: el álgebra relacional y el cálculo relacional. La manipulación lógica de los datos también hace factible la creación de lenguajes de interrogación más accesibles para un usuario no especialista en programación.

El trabajo publicado por Codd (1970), presentaba un nuevo modelo de datos que perseguía una serie de objetivos, que se pueden resumir en los siguientes.

Independencia física: el modo en el que se almacenan los datos no influya en su manipulación lógica y, por tanto, los usuarios que acceden a esos datos no tienen que modificar sus programas por cambios en el almacenamiento físico.

Independencia lógica: esto es, que el añadir, eliminar o modificar objetos de la base de datos no repercuta en los programas y/o usuarios que están accediendo a subconjuntos parciales de los mismos (vistas).

Flexibilidad: en el sentido de poder presentar a cada usuario los datos de la forma en que éste prefiera.

Uniformidad: las estructuras lógicas de los datos presentan un aspecto uniforme, lo que facilita la concepción y manipulación de la base de datos por parte de los usuarios.

Sencillez: las características anteriores, así como unos lenguajes de usuario muy sencillos, producen como resultado que el modelo de datos relacional sea fácil de comprender y de utilizar por parte del usuario final.

Para conseguir los objetivos citados, Codd introduce el concepto de "relación" (tabla) como una estructura básica del modelo. Todos los datos de la BD se representan en forma de relaciones cuyo contenido varía en el tiempo.

2.2.2. Modelo Relacional

El modelo relacional se basa en dos ramas de las matemáticas: la teoría de conjuntos y la lógica de predicados de primer orden. El hecho de que el modelo relacional esté basado en la teoría de las matemáticas es lo que lo hace tan seguro y robusto. Al mismo tiempo, estas ramas de las matemáticas

proporcionan los elementos básicos necesarios para crear una base de datos relacional con una buena estructura, y proporcionan las líneas que se utilizan para formular buenas metodologías de diseño.

Hay quien ofrece una cierta resistencia a estudiar complicados conceptos matemáticos para tan sólo llevar a cabo una tarea bastante concreta. Es habitual escuchar quejas sobre que las teorías matemáticas en las que se basa el modelo relacional y sus metodologías de diseño, no tienen relevancia en el mundo real o que no son prácticas. No es cierto: las matemáticas son básicas en el modelo relacional. Pero, por fortuna, no hay que aprender teoría de conjuntos o lógica de predicados de primer orden para utilizar el modelo relacional.

La teoría matemática proporciona la base para el modelo relacional y, por lo tanto, hace que el modelo sea predecible, fiable y seguro. La teoría describe los elementos básicos que se utilizan para crear una base de datos relacional y proporciona las líneas a seguir para construirla. El organizar estos elementos para conseguir el resultado deseado es lo que se denomina diseño.

2.2.2.1. Atributos y Dominios

Un atributo es el nombre de una columna de una relación. En el modelo relacional, las relaciones se utilizan para almacenar información sobre los objetos que se representan en la base de datos. Una relación se representa gráficamente como una tabla bidimensional en la que las filas corresponden a registros individuales y las columnas corresponden a los campos o atributos de esos registros. Los atributos pueden aparecer en la relación en cualquier orden.

Por ejemplo, la información de los centros de trabajo de una empresa se representa mediante la relación CENTROS_TRABAJO, que tiene columnas para los atributos CodCentro (número de Centro de Trabajo), Dirección, Población, Provincia, Teléfono y Fax. La información sobre los empleados se

representa mediante la relación EMPLEADOS, que tiene columnas para los atributos CodEmpledo (número de empleado que se su DNI), Nombre, Dirección, Población, Teléfono, Puesto, Centro (número del centro de trabajo al que pertenece el empleado). A continuación se muestra una instancia de la relación CENTRO_TRABAJO y una instancia de la relación EMPLEADOS. Como se puede observar, cada columna contiene valores de un solo atributo. Por ejemplo, la columna CodCentro sólo contiene números de Centros de trabajo que existen.

CENTROS_TRABAJO

CodEmpresa	**Dirección**	**Población**	**Provincia**	**Teléfono**	**Fax**
0004	Goya, 35	Madrid	Madrid	915669980	915669981
0006	Castilla, 5	Madrid	Madrid	915059900	915059856
0008	Esperanza,1	Córdoba	Córdoba	957345790	957345794
0001	Madrid	Getafe	Madrid	919856748	919856740
0003	Cantabria,45	Sitges	Barcelona	934578732	934578730

EMPLEADOS

CodEmpleado	**Nombre**	**Dirección**	**Población**	**Teléfono**	**Puesto**	**Centro**
09485847T	Aranch a Perez	Garcilaso	Getade	916947865	Administrativo	0001
88987664F	Pedro García	Castilla, 5	Madrid	915673898	Gerente área	0004
38883483E	Elena Gomez	Gibraltar	Madrid	913456722	Comercial	0004

28837488w	Ignacio Casado	Madrid	Barcelona	934765678	Obrero	0003
00039433R	Ana Ruiz	Cantabria, 45	Tarrasa	938876577	Contable	0003

Un dominio es el conjunto de valores legales de uno o varios atributos. Los dominios constituyen una poderosa característica del modelo relacional. Cada atributo de una base de datos relacional se define sobre un dominio, pudiendo haber varios atributos definidos sobre el mismo dominio. La siguiente tabla muestra los dominios de los atributos de la relación CENTROS_TRABAJO.

Nótese que en esta relación hay dos atributos que están definidos sobre el mismo dominio, Teléfono y Fax.

Atributo	**Nombre del dominio**	**Descripción**	**Definición**
CodCentro	CENTRO	Posibles valores de número de Centros de trabajo	4 caracteres
Direccion	DIRECCION	Nombres y número de calles de España	50 Caracteres
Población	NOM_LUGAR	Nombres de las poblaciones o provincias de España	15 caracteres
Provincia	NOM_LUGAR	Nombres de las poblaciones o provincias de España	15 caracteres
Telefono	NUM_TEL_FAX	Números de teléfono de España	9 caracteres
Fax	NUM_TEL_FAX	Números de teléfono de España	9 caracteres

El concepto de dominio es importante porque permite que el usuario defina, en un lugar común, el significado y la fuente de los valores que los atributos pueden tomar. Esto hace que haya más información disponible para el sistema cuando éste va a ejecutar una operación relacional, de modo que las operaciones que son semánticamente incorrectas, se pueden evitar. Los SGBD relacionales no ofrecen un soporte completo de los dominios ya que su implementación es extremadamente compleja.

Una tupla es una fila de una relación. Los elementos de una relación son las tuplas o filas de la tabla. En la relación CENTROS_TRABAJO, cada tupla tiene seis valores, uno para cada atributo. Las tuplas de una relación no siguen ningún orden.

El grado de una relación es el número de atributos que contiene. La relación CENTROS_TRABAJO es de grado seis porque tiene seis atributos. Esto quiere decir que cada fila de la tabla es una tupla con seis valores. El grado de una relación no cambia con frecuencia.

La cardinalidad de una relación es el número de tuplas que contiene. Ya que en las relaciones se van insertando y borrando tuplas a menudo, la cardinalidad de las mismas varía constantemente.

2.2.2.2. Relaciones

Una relación es una tabla con columnas y filas. Un SGBD sólo necesita que el usuario pueda percibir la base de datos como un conjunto de tablas. Esta percepción sólo se aplica a la estructura lógica de la base de datos (en el nivel externo y conceptual de la arquitectura de tres niveles ANSI-SPARC). No se aplica a la estructura física de la base de datos, que se puede implementar con distintas estructuras de almacenamiento.

Definición formal:

Una relación R definida sobre un conjunto de dominios D1, D2, ... Dn consta de:

Cabecera: conjunto fijo de pares atributo:dominio

Donde cada atributo Aj corresponde a un único dominio Dj y todos los Aj son distintos, es decir, no hay dos atributos que se llamen igual. El grado de la relación R es n.

Cuerpo: conjunto variable de tuplas. Cada tupla es un conjunto de pares atributo:valor con, i = 1, 2, ... m donde m es la cardinalidad de la relación R. En cada par (Aj : vij) se tiene que vij Î Dj

La relación CENTROS_TRABAJO tiene la siguiente cabecera:

{ (CodCentro:CENTRO), (Dirección:DIRECCION), (Poblacion:NOM_LUGAR), (Provincia:NOM_LUGAR), (Teléfono:NUM_TEL_FAX), (Fax:NUM_TEL_FAX)}.

Siendo la siguiente una de sus tuplas:

{ (CodCentro:0004), (Dirección:Goya, 35), (Poblacion:Madrid), (Provincia:Madrid), (Teléfono: 915669980), (Fax: 915669981)}.

Este conjunto de pares no está ordenado, por lo que esta tupla y la siguiente, son la misma:

{ (Provincia:Madrid), (CodCentro:0004), (Poblacion:Madrid), , (Fax: 915669981) , (Dirección:Goya, 35), (Teléfono: 915669980) }.

Las relaciones tienen las siguientes características:

- Cada relación tiene un nombre y éste es distinto del nombre de todas las demás.

- Los valores de los atributos son atómicos: en cada tupla, cada atributo toma un solo valor. Se dice que las relaciones están normalizadas.

- No hay dos atributos que se llamen igual.

- El orden de los atributos no importa: los atributos no están ordenados.
- Cada tupla es distinta de las demás: no hay tuplas duplicadas.

- El orden de las tuplas no importa: las tuplas no están ordenadas

2.2.2.3. Claves

Ya que en una relación no hay tuplas repetidas, éstas se pueden distinguir unas de otras, es decir, se pueden identificar de modo único. La forma de identificarlas es mediante los valores de sus atributos.

Una superclave es un atributo o un conjunto de atributos que identifican de modo único las tuplas de una relación.

Una clave candidata es una superclave en la que ninguno de sus subconjuntos es una superclave de la relación. El atributo o conjunto de atributos de la relación es una clave candidata para sí y sólo si satisface las siguientes propiedades:

- *Unicidad:* nunca hay dos tuplas en la relación con el mismo valor.

- *Irreducibilidad (minimalidad):* ningún subconjunto de tiene la propiedad de unicidad, es decir, no se pueden eliminar componentes de sin destruir la unicidad.

Cuando una clave candidata está formada por más de un atributo, se dice que es una clave compuesta. Una relación puede tener varias claves candidatas.

Por ejemplo, en la relación CENTROS_TRABAJO, el atributo Población no es una clave candidata ya que puede haber varios Centros de Trabajo en una misma población. Sin embargo, ya que la empresa asigna un código único a cada Centro de Trabajo, el atributo CodCentro si es una clave candidata de la relación CENTROS_TRABAJO. También son claves candidatas de esta relación los atributos Teléfono y Fax.

Para identificar las claves candidatas de una relación no hay que fijarse en un estado o instancia de la base de datos. El hecho de que en un momento dado no haya duplicados para un atributo o conjunto de atributos, no garantiza que los duplicados no sean posibles. Sin embargo, la presencia de duplicados en un estado de la base de datos sí es útil para demostrar que cierta combinación de atributos no es una clave candidata.

El único modo de identificar las claves candidatas es conociendo el significado real de los atributos, ya que esto permite saber si es posible que aparezcan duplicados. Sólo usando esta información semántica se puede saber con certeza si un conjunto de atributos forman una clave candidata. Por ejemplo, viendo la instancia anterior de la relación EMPLEADOS se podría pensar que el atributo Nombre es una clave candidata. Pero ya que este atributo es el nombre completo de un empleado y es posible que haya dos empleados con el mismo, el atributo no es una clave candidata.

La clave primaria de una relación es aquella clave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria. En el peor caso, la clave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función.

Las claves candidatas que no son escogidas como clave primaria son denominadas claves alternativas.

Una clave ajena es un atributo o un conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (puede ser la misma). Las claves ajenas representan relaciones entre datos. El atributo CodEmpleado de EMPLEADOS relaciona a cada empleado con el Centro de Trabajo al que pertenece. Este atributo es una clave ajena cuyos valores hacen referencia al atributo CodCentro, clave primaria de CENTROS_TRABAJO. Se dice que un valor de clave ajena representa una referencia a la tupla que contiene el mismo valor en su clave primaria (tupla referenciada).

2.2.2.4. Esquema Relacional

Una base de datos relacional es un conjunto de relaciones normalizadas. Para representar el esquema de una base de datos relacional se debe dar el nombre de sus relaciones, los atributos de éstas, los dominios sobre los que se definen estos atributos, las claves primarias y las claves ajenas.

El esquema de la base de datos de la empresa inmobiliaria es el siguiente:

CENTROS_TRABAJO	(CodCentro, Dirección, Área, Población, Teléfono, Fax)
EMPLEADOS	(CodEmpleado, Nombre, Dirección,Población, Teléfono, Puesto, codCentro)
INMUEBLE	(Inum, Calle, Area, Población, Tipo, Hab, Alquiler, Pnum, Enum, CodCentro)
INQUILINO	(Qnum, Nombre, Apellido, Dirección, Teléfono, Tipo_pref, Alquiler_max)
PROPIETARIO	(Pnum, Nombre, Apellido, Dirección, Teléfono)
VISITA	(Qnum, Inum, Fecha, Comentario)

En el esquema, los nombres de las relaciones aparecen seguidos de los nombres de los atributos encerrados entre paréntesis. Las claves primarias son los atributos subrayados. Las claves ajenas se representan mediante los siguientes diagramas referenciales:

EMPLEADOS	CodCentro →	CENTROS_TRABAJO	:	*Centro que pertenece el empleado.*
INMUEBLE	Pnum →	PROPIETARIO	:	*Propietario del inmueble.*
INMUEBLE	Enum →	EMPLEADOS	:	*Empleado encargado del inmueble.*
INMUEBLE	CodCentro →	OFICINA	:	*Oficina a la que pertenece el inmueble.*
VISITA	Qnum →	INQUILINO	:	*Inquilino que ha visitado el inmueble.*
VISITA	Inum →	INMUEBLE	:	*Inmueble que ha sido visitado.*

A continuación se muestra un estado (instancia) de la base de datos cuyo esquema se acaba de definir:

CENTROS_TRABAJO

CodEmpresa	Dirección	Población	Provincia	Teléfono	Fax
0004	Goya, 35	Madrid	Madrid	915669980	915669981
0006	Castilla, 5	Madrid	Madrid	915059900	915059856
0008	Esperanza,1	Córdoba	Córdoba	957345790	957345794
0001	Madrid	Getafe	Madrid	919856748	919856740
0003	Cantabria,45	Sitges	Barcelona	934578732	934578730

EMPLEADOS

CodEmpleado	Nombre	Dirección	Población	Teléfono	Puesto	Centro
09485847T	Aranch a Perez	Garcilaso	Getade	916947865	Administrativo	0001
88987664F	Pedro García	Castilla, 5	Madrid	915673898	Gerente área	0004
38883483E	Elena Gomez	Gibraltar	Madrid	913456722	Comercial	0004
28837488w	Ignacio Casado	Madrid	Barcelona	934765678	Obrero	0003
00039433R	Ana Ruiz	Cantabria, 45	Tarrasa	938876577	Contable	0003

INMUEBLE

Inum	Calle	Area	Población	Tipo	Hab	Alquiler	Pnum	CodCentro
IA14	Enmedio, 128	Centro	Madrid	Casa	6	600	P46	0004
IL94	Riu Ebre, 24	Ronda Sur	Barcelona	Piso	4	350	P87	0003
IG4	Sorell, 5	Grao	Madrid	Piso	3	300	P40	0001
IG36	Alicante, 1		Madrid	Casa	3	325	P93	0004
IG21	San Francisco, 10		Barcelona	Piso	5	550	P87	0003

PROPIETARIO

Pnum	Nombre	Apellido	Dirección	Teléfono
P46	Amparo	Felip	Asensi 24, Madrid	964 230 680
P87	Manuel	Obiol	Av. Libertad 15, Barcelona	964 450 760
P40	Alberto	Estrada	Av. del Puerto 52, Madrid	964 200 740
P93	Yolanda	Robles	Purísima 4, Madrid	964 710 430

INQUILINO

Qnum	Nombre	Apellido	Dirección	Teléfono	Tipo	Alquiler
Q76	Juan	Felip	Barceló 47, Barcelona	964 282 540	Piso	375
Q56	Ana	Grangel	San Rafael 45, Madrid	964 551 110	Piso	300
Q74	Elena	Abaso	Navarra 76, Gijón	964 205 560	Casa	700
Q62	Alicia	Mori	Alloza 45, Córdoba	964 229 580	Piso	550

VISITA

Qnum	Inum	Fecha	Comentario
Q56	IA14	24/11/99	Muy pequeño
Q76	IG4	20/10/99	Poco luminoso
Q56	IG4	26/11/99	
Q62	IA14	14/11/99	no tiene salón
Q56	IG36	28/10/99	

2.2.2.5. Reglas de Integridad

Una vez definida la estructura de datos del modelo relacional, pasamos a estudiar las reglas de integridad que los datos almacenados en dicha estructura deben cumplir para garantizar que son correctos.

Al definir cada atributo sobre un dominio se impone una restricción sobre el conjunto de valores permitidos para cada atributo. A este tipo de restricciones se les denomina restricciones de dominios. Hay además una regla de integridad muy importante que son restricciones que se deben cumplir en todas las bases de datos relacionales y en todos sus estados o instancias (las reglas se deben cumplir todo el tiempo). Esta regla es la regla de integridad referencial. Antes de definirla, es preciso conocer el concepto de nulo.

- Nulos

Cuando en una tupla un atributo es desconocido, se dice que es nulo. Un nulo no representa el valor cero ni la cadena vacía, éstos son valores que tienen significado. El nulo implica ausencia de información, bien porque al insertar la tupla se desconocía el valor del atributo, o bien porque para dicha tupla el atributo no tiene sentido.

Ya que los nulos no son valores, deben tratarse de modo diferente, lo que causa problemas de implementación. De hecho, no todos los SGBD relacionales soportan los nulos.

- Regla de integridad referencial

La segunda regla de integridad se aplica a las claves ajenas: si en una relación hay alguna clave ajena, sus valores deben coincidir con valores de la clave primaria a la que hace referencia, o bien, deben ser completamente nulos.

La regla de integridad referencial indica lo que es un estado ilegal, pero no dice cómo puede evitarse. Existen dos opciones: rechazar la operación, o bien aceptar la operación y realizar operaciones adicionales compensatorias que conduzcan a un estado legal.

2.2.2.6. Las 12 normas de Codd

Para valorar que SGBDR seguían el Modelo Relacional y cuáles no, en 1985 Codd propone 12 reglas que todos ellos deben seguir:

1. Representación de la Información: representación única y a nivel lógico.
2. Acceso garantizado: a una relación (por su nombre), a sus tuplas (por la clave),o a sus atributos (a través de los nombres de atributo)
3. Tratamiento sistemático de valores nulos.
4. Catálogo relacional: la meta información se almacena en la BD.
5. Sublenguaje de Datos completo: permite describir relaciones, vistas, restricciones, permisos (acceso), manejar datos, y gestionar transacciones.
6. Actualización de Vistas: toda vista actualizable de poder ser actualizada.
7. Actualizaciones y Recuperaciones de alto nivel: a nivel de conjuntos.
8. Independencia física de los datos: datos independientes de los caminos físicos.
9. Independencia lógica de los datos: datos independientes de los procesos.
10. Independencia de la integridad: la semántica no depende del SGBD.
11. Independencia de la distribución: soportará BD distribuidas.
12. Reglas de no Subversión: si el SGBD soporta un lenguaje de bajo nivel, este no podrá usarse nunca para ignorar las reglas de integridad.

2.2.3. Modelos Postrelacionales

La necesidad de almacenar y manipular datos complejos que no son fáciles de modelar usando las simples tablas del modelo relacional, así como el desarrollo de lenguajes de programación orientados a objetos, permitieron el desarrollo de otros modelos de datos.

2.2.3.1. Modelo Orientado a objetos

El modelo orientado a objetos fue desarrollado en los 90s para manejar los datos requeridos para aplicaciones avanzadas como sistemas de información geográfica (GIS), multimedia, diseño asistido por computadora y manufactura asistida por computadora (CAD/CAM), y otros ambientes complejos. El modelo orientado a objetos es un modelo semántico similar al modelo entidad/relación. Éste extiende los conceptos del E/R agregando encapsulación, que implica incorporar tanto datos como funciones en una unidad donde son protegidos de modificaciones del exterior. A diferencia del E/R con entidades y atributos, este modelo usa objetos que tienen estado y comportamiento. El estado de un objeto se determina por los valores de sus atributos (variables de instancias). El comportamiento es el conjunto de métodos (funciones) definidas para el objeto.

Para crear una BD orientada a objetos, El diseñador comienza definiendo clases, las cuales especifican los atributos y métodos para un conjunto de objetos. Cada objeto es entonces creado instanciando la clase, usando uno de sus propios métodos llamados constructores. La estructura de los objetos puede ser muy compleja. Cada objeto en una BD debe tener un identificador único que funciona como una llave primaria permanente, pero que no toma valores de ninguno de los atributos del objeto. Las clases que se relacionan con otras se agrupan para formar jerarquías de clases. Una diferencia importante entre los objetos de programa y los objetos de la BD es la persistencia, ya que los objetos de programa existen sólo mientras el programa está en ejecución, mientras que un objeto de BD permanece en existencia después de que la ejecución del programa se completa.

2.2.3.2. Modelo Objeto-Relacional

Algunos RDBMSs como Oracle, agregaron algunas capacidades orientadas a objetos a sus productos, resultando en bases de datos híbrida objeto-relacionales.El modelo objeto-relacional extiende al modelo relacional agregando algunos tipos de datos complejos y métodos. En vez de atomicidad y valores simples en los atributos que requiere el modelo relacional, este modelo permite atributos estructurados y tienen conjuntos de arreglos de valores. También permite métodos y herencia. El lenguaje SQL fue extendido en 1999 para crear y manipular los tipos de datos más complejos que soporta este modelo. Sin embargo, el lenguaje usado para manejar BD objeto-relacionales es más cercano al tipo de lenguaje para BD relacionales que el que se usa para BD estrictamente orientadas a objetos.

2.2.3.3. Modelo de Datos Semiestructurados

El amplio uso de Internet ha tenido gran impacto en el desarrollo de BD. Internet conecta a los usuarios a una amplia red de BDs en constante expansión, proporcionando acceso a bibliotecas digitales, recursos multimedia, recursos educativos y mucho más. Los sitios de comercio electrónico permiten acceso a BDs de información sobre productos y servicios a los clientes a través de mundo. Los dispositivos de cómputo inalámbrico y los clientes pequeños como PDAs son otros desarrollos que permiten a los usuarios conectarse a recursos de BDs en formas nuevas y flexibles.La mayoría de los modelos requiere que los tipos de entidades (u objetos o registros, dependiendo del modelo) tengan la misma estructura. La estructura se define en el esquema y permanece sin modificaciones a menos que el DBA cambie el esquema. En contraste, el modelo semi-estructurado permite una colección de nodos, cada uno conteniendo datos, posiblemente con diferentes esquemas. El nodo contiene información sobre la estructura de su contenido. Las BD semi-estructuradas son especialmente útiles cuando existen BD que tienen

diferentes esquemas integrados. Las BD individuales pueden verse como documentos y pueden agregarse etiquetas XML (Extensible Markup Language) a cada documento para describir su contenido. XML es un lenguaje similar a HTML (Hipertext Markup Language), pero es usado como un estándar para intercambio de datos en vez de presentación de datos. Las etiquetas XML se usan para identificar elementos, sub-elementos y atributos que almacenan datos. El esquema puede especificarse usando DTD (Document Type Definition) o por un esquema XML que identifica los elementos, sus atributos y sus relaciones con otros.

2.2.3.4. Data warehouses y minería de datos

Los data warehouses se desarrollaron en los 90s para proporcionar un método de captura de datos consolidando de muchas bases de datos. Un data warehouse generalmente almacena datos históricos de una organización, con el propósito de hacer minería de datos, que es un proceso de análisis de datos estadísticos para permitir a las organizaciones analizar sus propios registros.

2.3. Bases de Datos Activas.

Se expondrán los conocimientos teóricos referentes a los sistemas de bases de datos activas.

2.3.1. Introducción

En muchas aplicaciones, la base de datos debe evolucionar independientemente de la intervención del usuario como respuesta a un suceso o una determinada situación. En los sistemas de gestión de bases de datos tradicionales (pasivas), la evolución de la base de datos se programa en el código de las aplicaciones, mientras que en los sistemas de gestión de

bases de datos activas esta evolución es autónoma y se define en el esquema de la base de datos.

El poder especificar reglas con una serie de acciones que se ejecutan automáticamente cuando se producen ciertos eventos, es una de las mejoras de los sistemas de gestión de bases de datos que se consideran de gran importancia desde hace algún tiempo. Mediante estas reglas se puede hacer respetar reglas de integridad, generar datos derivados, controlar la seguridad o implementar reglas de negocio. De hecho, la mayoría de los sistemas relacionales comerciales disponen de disparadores (triggers). Se ha hecho mucha investigación sobre lo que debería ser un modelo general de bases de datos activas desde que empezaron a aparecer los primeros disparadores. El modelo que se viene utilizando para especificar bases de datos activas es el modelo evento–condición–acción.

Mediante los sistemas de bases de datos activas se consigue un nuevo nivel de independencia de datos: la independencia de conocimiento. El conocimiento que provoca una reacción se elimina de los programas de aplicación y se codifica en forma de reglas activas.

De este modo, al encontrarse las reglas definidas como parte del esquema de la base de datos, se comparten por todos los usuarios, en lugar de estar replicadas en todos los programas de aplicación. Cualquier cambio sobre el comportamiento reactivo se puede llevar a cabo cambiando solamente las reglas activas, sin necesidad de modificar las aplicaciones.

Además, mediante los sistemas de bases de datos activas se hace posible el integrar distintos subsistemas (control de accesos, gestión de vistas, etc.) y se extiende el ámbito de aplicación de la tecnología de bases de datos a otro tipo de aplicaciones.

Uno de los problemas que ha limitado el uso extensivo de reglas activas, a pesar de su potencial para simplificar el desarrollo de bases de datos y de aplicaciones, es el hecho de que no hay técnicas fáciles de usar para diseñar,

escribir y verificar reglas. Por ejemplo, es bastante difícil verificar que un conjunto de reglas es consistente, es decir, que no se contradice.

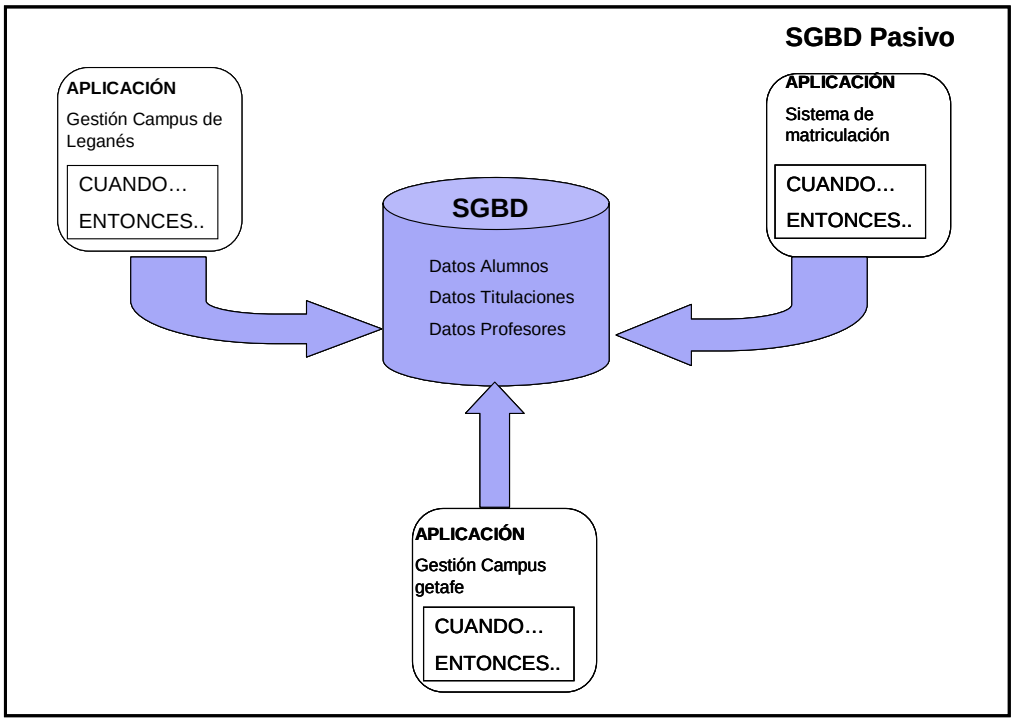
También es difícil garantizar la terminación de un conjunto de reglas bajo cualquier circunstancia. Para que las reglas activas alcancen todo su potencial, es necesario desarrollar herramientas para diseñar, depurar y monitorizar reglas activas que puedan ayudar a los usuarios en el diseño y depuración de sus reglas.

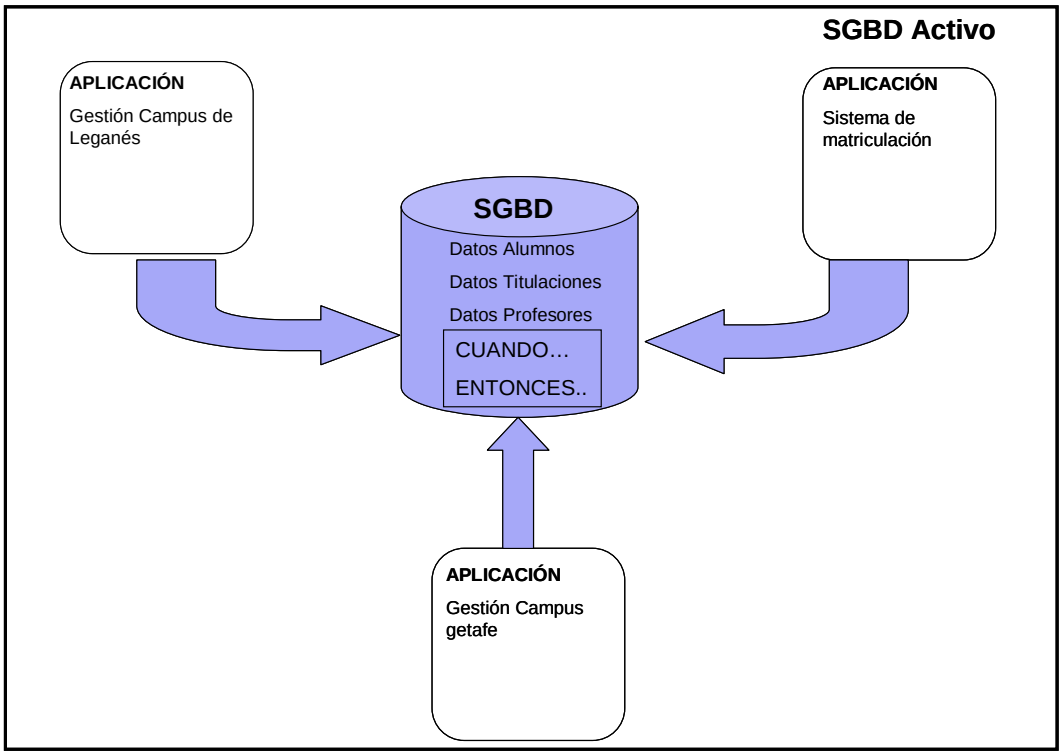
Las BBDD convencionales se consideran pasivas, ya que los usuarios o los programas son quienes deciden qué acciones llevar a cabo sobre éstas. En éstos casos la evolución de la base de datos se programa en las aplicaciones, esto implica que el 'Conocimiento' sea independiente de los datos, lo cual conlleva el riesgo de que unas aplicaciones tengan que tener mucha información de otras. Por ello surgen las BBDD activas, a cuyo término dio origen Morgenstern en 1983. Estas permiten que su evolución sea autónoma y esté definida en el propio esquema de la base de datos.

Se puede definir el comportamiento activo como la capacidad de un sistema de desarrollo de reaccionar, de forma autónoma, ante determinadas condiciones previamente definidas. De ésta forma, el conocimiento que permite llevar acabo unas determinas acciones, se independiza de la aplicación, de tal manera que es común a todas las aplicaciones que interactúen con la BBDD, sin necesidad de replicar información.

Por lo tanto en un comportamiento activo, tanto la detección de la situación, como la ejecución de la acción que esta desencadena, tienen lugar en el sistema de desarrollo.

Las siguientes figuras representan ambos comportamientos:





Las ventajas de este funcionamiento son:

- Mayor productividad
- Mejor rendimiento
- Reutilización de código
- Reducción del tráfico de mensajes
- Simplifican las aplicaciones, liberándolas de las características muy intrínsecas a la semántica de los datos.
- Mejor mantenimiento y reutilización de código, puesto que todo el conocimiento está centralizado en el SGBD.
- Facilidad de acceso a la BD de usuarios finales, ya que la integridad de la ésta es preservada por el propio SGBD, independientemente del medio de acceso.

Las propiedades deseables de un SGBD activo son:

- Proporcionar la funcionalidad de un SGBD pasivo sin decrementar el rendimiento en las tareas más frecuentes.
- Contener un mecanismo que permita a los usuarios y aplicaciones especificar el comportamiento activo deseado, y hacer que éste forme parte de la BD.
- Proporcionar utilidades de diseño y depuración de BD para comportamiento activo similares a los proporcionadas por los SGBD convencionales.
- El código incluido en la BD deberá tener el mismo nivel de expresividad, portabilidad, estandarización y facilidades de depuración que el código realizado en un 3GL. Su nivel de eficiencia deberá ser comparable al código equivalente ejecutado desde el sistema operativo.

Por ejemplo: Para una aplicación de inventario de una fábrica que interactúa con una base de datos de los productos que almacena la existencia de los mismos y sus necesidades mínimas y máximas, lo deseable sería que existiese un comportamiento activo en la BBDD, que desencadenase determinadas acciones cuando para un producto no se alcanzase el stock mínimo, o bien cuando se sobrepasasen del máximo las existencias del mismo.

Uno de los problemas que ha limitado el uso extensivo de éstos sistemas, a pesar de todo el gran potencial que ofrecen, es el hecho de que no hay técnicas fáciles para diseñar y verificar las acciones que se toman sobre determinados eventos.

Por lo tanto, la inclusión de conocimiento en un SGBD, permite que éstos cada vez asuman tareas más complejas, además facilita los siguientes mecanismos:
Modelo de conocimiento: es el mecanismo que permite describir la situación y la reacción correspondiente.
Modelo de ejecución: permite realizar el seguimiento de la situación y gestionar el comportamiento activo.

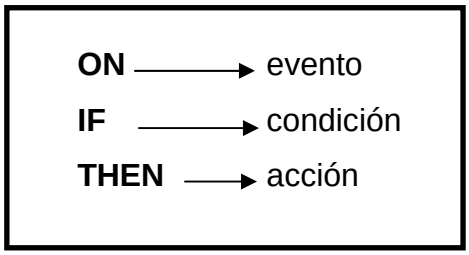
2.3.2. El modelo evento-condición-acción

Un sistema de bases de datos activas es un sistema de gestión de bases de datos (SGBD) que contiene un subsistema que permite la definición y la gestión de reglas de producción (reglas activas). Las reglas siguen el modelo evento–condición–acción (modelo ECA): cada regla reacciona ante un determinado evento, evalúa una condición y, si ésta es cierta, ejecuta una acción. La ejecución de las reglas tiene lugar bajo el control de un subsistema autónomo, denominado motor de reglas, que se encarga de detectar los eventos que van sucediendo y de planificar las reglas para que se ejecuten.

Los SGBD emplean reglas para especificar los mecanismos de situación-reacción.

El modelo que se ha empleado para especificar las reglas de las bases de datos activas se denomina **Evento-Condición-Acción (ECA).**

El formato genérico de estas reglas es:



Una regla en el modelo ECA tiene tres componentes:

- Evento (ó eventos) que activa la regla: pueden ser operaciones de consulta o actualización que se aplican explícitamente sobre la base de datos. También pueden ser eventos temporales (por ejemplo, a una determinada hora del día) u otro tipo de eventos externos (definidos por el usuario). Se puede concluir que un evento describe el suceso que

- Condición: determina si la acción de la regla se debe ejecutar. Una vez ocurre el evento disparador, se puede evaluar una condición (es opcional). Si no se especifica condición, la acción se ejecutará cuando suceda el evento. Si se especifica condición, la acción se ejecutará sólo si la condición se evalúa a verdadero.

- Acción: suele ser una secuencia de sentencias SQL, pero también puede ser una transacción de base de datos o un programa externo que se ejecute automáticamente.

Casi todos los sistemas relacionales incorporan reglas activas simples denominadas disparadores (triggers), que están basados en el modelo ECA:

- Los eventos son sentencias SQL de manejo de datos (INSERT, DELETE, UPDATE).
- La condición (que es opcional) es un predicado booleano expresado en SQL.

- La acción es un secuencia de sentencias SQL, que pueden estar inmersas en un lenguaje de programación integrado en el producto que se esté utilizando (por ejemplo, PL/SQL en Oracle).

El modelo ECA se comporta de un modo simple e intuitivo: cuando ocurre el evento, si la condición es verdadera, entonces se ejecuta la acción. Se dice que el disparador es activado por el evento, es considerado durante la verificación de su condición y es ejecutado si la condición es cierta. Sin embargo, hay diferencias importantes en el modo en que cada sistema define la activación, consideración y ejecución de disparadores.

Los disparadores relacionales tienen dos niveles de granularidad: a nivel de fila y a nivel de sentencia. En el primer caso, la activación tiene lugar para cada tupla involucrada en la operación y se dice que el sistema tiene un comportamiento orientado a tuplas. En el segundo caso, la activación tiene lugar sólo una vez para cada sentencia SQL, refiriéndose a todas las tuplas invocadas por la sentencia, con un comportamiento orientado a conjuntos. Además, los disparadores tienen funcionalidad inmediata o diferida. La evaluación de los disparadores inmediatos normalmente sucede inmediatamente después del evento que lo activa (opción después), aunque también puede precederlo (opción antes) o ser evaluados en lugar de la ejecución del evento (opción en lugar de). La evaluación diferida de los disparadores tiene lugar al finalizar la transacción en donde se han activado (tras la sentencia COMMIT).

Un disparador puede activar otro disparador. Esto ocurre cuando la acción de un disparador es también el evento de otro disparador. En este caso, se dice que los disparadores se activan en cascada.

Las reglas se describen usando un lenguaje y pueden ser de dos tipos:

- Reglas CA o de producción: sin reglas que no tienen evento
- Reglas EA o Trigger: Son reglas sin condición.

2.3.2.1. Las reglas activas

Diseño de Eventos

Especificar un evento consiste en dar una descripción del acontecimiento que se pretende monitorizar. La naturaleza de la descripción y la forma en la que el evento se puede detectar depende de las capacidades del sistema. El diseño de los eventos comprende los siguientes aspectos:

- Fuente del evento: La ocurrencia del evento puede ser ocasionada por diferentes tipos de sucesos, tales como: Modificación de datos, recuperación de datos, gestión de transacciones, envío de mensajes, excepciones, tiempo, etc.

- Tipo de evento: los eventos pueden ser primitivos o compuestos. Para poder combinarlos existen operadores de un Álgebra de eventos.
  - Operadores lógicos (Disyunción, conjunción, etc.)
  - Secuencia
  - Compasiones temporales (la primera vez en un intervalo, ó n veces en el mismo intervalo, etc.)

- Granuralidad del evento: son los cambios que considera la ocurrencia de un evento.
  - Para los sistemas relaciones: existen eventos que consideran una sola tupla (row triggers) ó eventos que consideran todas las tuplas (statement triggers).
  - Para los sistemas Orientados a Objetos: existen eventos que consideran una sola instancia, u otros que consideran todas las instancias de una clase.

- Instante de ejecución respecto al comando: es el momento en el que se dispara la acción con respecto al evento que los produce y puede ser:
  - Before: antes de que se produzca el evento
  - After: después de que se produzca el evento

- Instead of: Se fuerza a que se realice otro curso de acción diferente (en lugar de ejecutar el evento).

Condiciones

Expresan una condición adicional a efectuar, una vez que la regla es disparada y antes de que se ejecute. Los valores devueltos en la evaluación de la condición podrían entregarse a la acción. El diseño de las condiciones comprende los siguientes aspectos:

- Amplitud de la condición: Es aquello acerca de los que se puede preguntar el la condición.
  - Predicados de BD:
  - Predicados de BD restringidos:
  - Consultas de BD
  - Procedimientos de aplicación
- Ámbito de la condición: es el entorno sobre el que se evalúa la condición.
  - Parámetros del evento ($Bind_E$)
  - Estado de la BD: al comienzo de la transacción, cuando ocurrió el evento ó cuando se evalúa la condición.

Acciones

Expresan el comportamiento deseado una vez que la regla se ha disparado y la condición se ha evaluado como verdadera. Existe la posibilidad de especificar varias acciones para la misma regla con un orden entre ellas.

El diseño de las acciones comprende los siguientes aspectos:

- Amplitud de la acción: determina aquellos que se puede ejecutar en la acción.
  - Operaciones de modificación
  - Operaciones de recuperación
  - Otros comandos de BD
  - Comandos de lenguajes extendidos de BD (por ejemplo PL/SQL)

- Procedimientos de aplicación
- Abortar la transacción

- Ámbito de la acción: es el entorno sobre el que se ejecuta la acción o aquello sobre lo que puede actuar
  - Parámetros del evento ($Bind_E$) ó de la condición ($Bind_c$)
  - Estado de la BD: comienzo de la transacción (BD_T), ocurrencia del evento (BD_E), evaluación de la condición (BD_c), ejecución de la acción (BD_A).

Tanto la evaluación de la condición, como la ejecución de la acción, se hacen con respecto a un cambio de estado de la BD, por lo tanto es necesario poder acceder a los valores de transición ($Bind_E$, $Bind_C$, BD_T, BD_E, BD_C, BD_A).

2.3.2.2. Procesamiento de reglas activas

Hay dos algoritmos alternativos para el procesamiento de las reglas activadas por una sentencia: al algoritmo iterativo y al algoritmo recursivo. Ambos se detallan a continuación:

Algoritmo Iterativo

mientras existan reglas activadas:

1. seleccionar una regla activada R
2. comprobar la condición de R
3. si la condición es cierta, ejecutar la acción de R

fin mientras

Algoritmo Recursivo

mientras existan reglas activadas:

1. seleccionar una regla activada R
2. comprobar la condición de R
3. si la condición es cierta
   3.1. ejecutar la acción de R

3.2. ejecutar este algoritmo para las reglas activadas por la acción de R

fin mientras

2.3.2.3. Caracterísitcas de las reglas activas

Además de las características que poseen los disparadores que incorporan los sistemas relacionales, algunos sistemas más avanzados y algunos prototipos de bases de datos activas ofrecen algunas características que incrementan la expresividad de las reglas activas:

- Respecto a los eventos, éstos pueden ser temporales o definidos por el usuario. Los eventos temporales permiten utilizar expresiones dependientes del tiempo, como por ejemplo: cada viernes por la tarde, a las 17:30 del 29/06/2002. Los eventos definidos por el usuario son eventos a los que el usuario da un nombre y que son activados por los programas de usuario. Por ejemplo, se podría definir el evento de usuario 'nivelalto-azufre' y que una aplicación lo activara; esto activaría la regla que reacciona al evento.

- La activación de los disparadores puede que no dependa de un solo evento sino que dependa de un conjunto de eventos relacionados en una expresión booleana que puede ser una simple disyunción o una combinación más compleja que refleje la precedencia entre eventos y la conjunción de eventos.

- La consideración y/o ejecución de reglas se puede retrasar. En este caso, la consideración y/o la ejecución tienen lugar durante transacciones distintas, que pueden ser completamente independientes o pueden estar coordinadas con la transacción en la que se ha verificado el evento.

- Los conflictos entre reglas que se activan por el mismo evento se pueden resolver mediante prioridades explícitas, definidas directamente

por el usuario cuando se crea la regla. Se pueden expresar como una ordenación parcial (utilizando relaciones de precedencia entre reglas), o como una ordenación total (utilizando prioridades numéricas).Las prioridades explícitas sustituyen a los mecanismos de prioridades implícitos que poseen los sistemas.

- Las reglas se pueden organizar en conjuntos y cada conjunto se puede habilitar y deshabilitar independientemente.

2.3.2.4. Propiedades de las reglas activas

No es difícil diseñar reglas activas de modo individual, una vez se han identificado claramente el evento, la condición y la acción. Sin embargo, entender el comportamiento colectivo de las reglas activas es más complejo ya que su interacción suele ser sutil. Por este motivo, el problema principal en el diseño de las bases de datos activas está en entender el comportamiento de conjuntos complejos de reglas. Las propiedades principales de estas reglas son terminación, confluencia e idéntico comportamiento observable.

- Un conjunto de reglas garantiza la **terminación** cuando, para cada transacción que puede activar la ejecución de reglas, esta ejecución produce un estado final en un número finito de pasos.

- Un conjunto de reglas garantiza la **confluencia** cuando, para cada transacción que puede activar la ejecución de reglas, la ejecución termina produciendo un estado final único que no depende del orden de ejecución de las reglas.

- Un conjunto de reglas garantiza un **comportamiento observable idéntico** cuando, para cada transacción que puede activar la ejecución de reglas, esta ejecución es confluyente y todas las acciones visibles llevas a cabo por la regla son idénticas y producidas en el mismo orden.

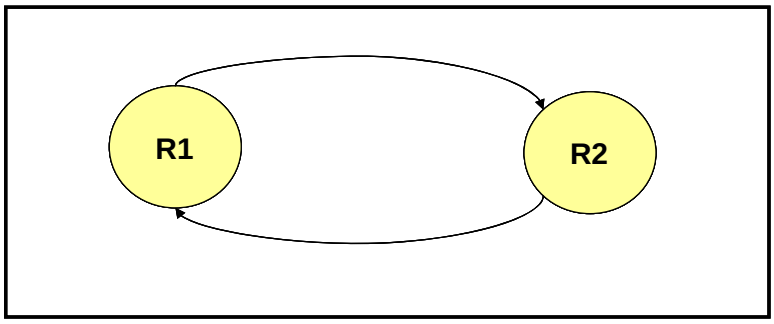
Estas propiedades no tienen la misma importancia. Concretamente, la terminación es una propiedad esencial; se debe evitar la situación en que las transacciones, activadas por el usuario, causan ejecuciones infinitas por la activación recursiva de reglas. Por otra parte, la confluencia y el idéntico comportamiento observable no son esenciales.

El proceso del análisis de reglas permite la verificación de si las propiedades deseadas se cumplen en un conjunto de reglas. Una herramienta esencial para verificar la terminación es el grafo de activación, que representa interacciones entre reglas. El grafo se crea incluyendo un nodo para cada regla y un arco de la regla R1 a la regla R2 cuando la acción de R1 contiene un sentencia del lenguaje de manejo de datos que es también uno de los eventos de R2. Una condición necesaria para la no terminación es la presencia de ciclos en el grafo de activación: sólo en este caso podemos tener una secuencia infinita de ejecución de reglas.

Los sistemas que tienen muchas reglas activas suelen ser cíclicos. Sin embargo, sólo unos pocos ciclos son los que provocan situaciones críticas. De hecho, el que un grafo sea cíclico es condición necesaria pero no suficiente para la no terminación.

Un simple ejemplo es:
R1: después de insertar en TABLA1, para cada fila, modificar a ‘V’ el valor de ATRIBUTO1 de TABLA2.
R2: después de modificar el valor de ATRIBUTO1 de TABLA2, para cada fila, insertar en TABLA1.

Resulta fácil ver en este ejemplo que estas dos reglas se pueden activar entre sí indefinidamente, lo que conduce a que no se llegue a su terminación. Sin embargo, si se escriben docenas de reglas es muy difícil determinar si garantizan la terminación, para lo cual es muy útil utilizar un *grafo de activación*, como el que se muestra a continuación:



Estos conflictos entre reglas que se activan por el mismo evento se pueden resolver mediante prioridades explícitas, definidas directamente por el usuario cuando se crea la regla. Se pueden expresar como una ordenación parcial (utilizando relaciones de precedencias entre reglas), o como una ordenación total (utilizando prioridades numéricas). Las prioridades explícitas sustituyen a los mecanismos de prioridades implícitos que poseen los sistemas.

2.3.3. El estándar de SQL y las bases de datos activas

El primer estándar de SQL donde se consideró la noción de ser activa de una base de datos fue en SQL:99 (es decir, en SQL 3); para el estándar que salió en 1992 no se incluyó este aspecto porque ya era muy extenso el estándar y quienes lo desarrollaron pensaron, equivocadamente, que los implementadores de DBMS no iban a implementar ese aspecto en los manejadores por un tiempo. Sin embargo, los desarrolladores sí implementaron este aspecto, pero como no había estándar, cada uno realizó su implementación independientemente, obteniendo así una gran diversidad de implementaciones de triggers, específicamente. La descripción de triggers y assertions sí se encuentra en SQL:99, por lo tanto, ahora se quiere tratar de uniformizar todas las implementaciones existentes.

En síntesis, los aspectos de base de datos activas en el estándar SQL:99 se concentran en los siguientes aspectos: constraints, triggers y assertions. A continuación se presenta una breve descripción de cada uno de estos aspectos:

- **Constraints**. Son especificaciones del DDL de SQL que se aplican a columnas o tuplas de una tabla. Algunos tipos especiales de constraints son: UNIQUE, NOT NULL, REFERENCES, CHECK; éste último permite especificar una amplia gama de reglas, como por ejemplo, rangos de valores y listas de valores, entre otros. Es conveniente que cada constraint tenga un nombre, pues cuando el constraint es violado el sistema indica su nombre y se puede saber exactamente qué falló.

  La verificación de las restricciones y el hacer que se cumplan se puede hacer de manera inmediata (constraint check time IMMEDIATE) durante la ejecución de la transacción o de forma diferida (constraint check time DEFERRED), es decir, a tiempo de compromiso de la transacción (commit).

  Los constraints pueden ser violados solo por una o más tuplas de una tabla y no por la tabla en sí misma, es decir, que una tabla vacía cumple con todos los constraints, una tabla vacía no viola restricción alguna.

- **Assertions.** Es un tipo de restricción especial que se puede especificar en SQL sin que deba estar asociada a una tabla en particular, como es el caso de los constraints. Generalmente se utilizan para describir restricciones que afectan a más de una tabla.

  Como los constraints sólo se pueden establecer sobre tuplas de una tabla, los assertions son útiles cuando es necesario especificar una condición general de la base de datos que no se puede asociar a una tabla específica de la base de datos. Por esta característica de poder especificar un assertion para toda la base de datos y no para una tabla en particular, también se les llama standalone constraints. Esto tiene grandes ventajas a nivel conceptual, pero las hace difíciles de implementar.

- **Triggers**. Los triggers en SQL:99 son parecidos a los constraints, pero proveen mayor flexibilidad pues el usuario puede especificar un conjunto de acciones complejas a ejecutarse cuando el trigger se dispare. Los constraints actúan cuando se viola lo que ellos indican, pero el usuario no puede especificar acción alguna con esa violación, sino que el DBMS simplemente envía un mensaje de error e impide que se realice la operación que produjo la violación.

  La descripción de un trigger contiene ocho (8) componentes, a saber:

  Nombre, debe ser único en el esquema donde se define.

  Tabla sujeto, es el nombre de la tabla cuyos datos, al ser alterados, van a activar al trigger.

  Tiempo de la acción, le dice al DBMS cuando ejecutar el cuerpo del trigger, lo cual puede ser antes (BEFORE) o después (AFTER) que ocurra el evento del trigger.

  Evento, indica cuál es el comando de actualización sobre la tabla, que va a activar el trigger, este evento puede ser uno o más de los siguientes: INSERT, DELETE y UPDATE.

  Lista de atributos, sólo se pueden especificar si el evento es un UPDATE.

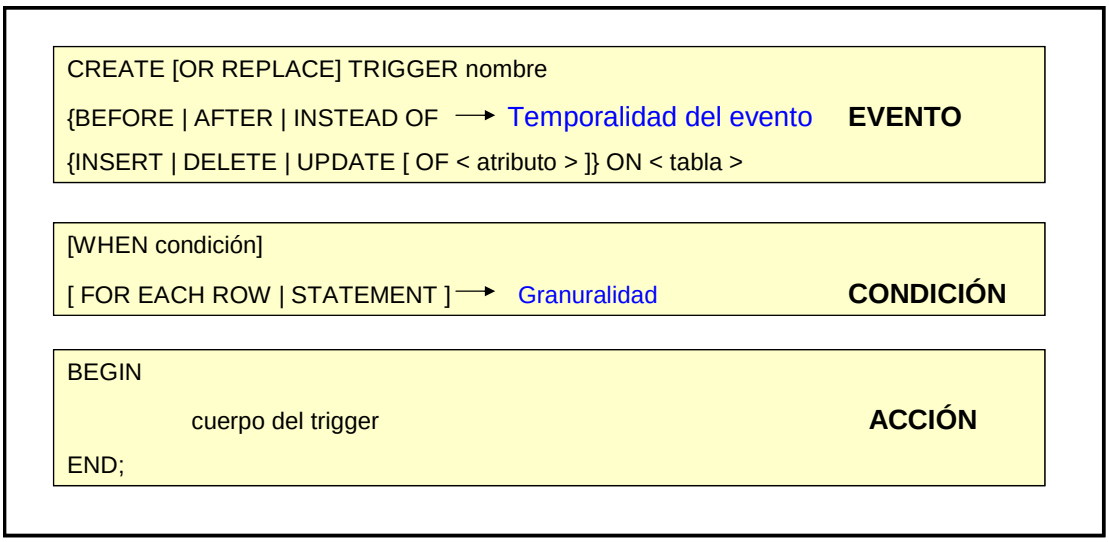
  Aliases, tanto los valores de la tupla antes y después de la actualización, como las tablas pueden tener nombres que fungen de sinónimos de los nombres originales.

  Cuerpo del trigger, comandos de SQL que se ejecutan cuando el trigger se dispara.

  Sello de tiempo, indica cuándo fue creado el trigger.

  Los triggers se definen con un comando de CREATE TRIGGER y para borrarlos se utiliza el comando DROP TRIGGER.

La estructura de un disparador es:



2.3.4. Algunas cuestiones de Diseño e implementación de BBDD activas

Además de crear reglas un sistema de bases de datos activas debería permitir que los usuarios activen, desactiven y eliminen reglas haciendo referencia a los nombres de éstas. Una regla desactivada no se activará no se activará por el evento disparador. Ésta característica permite que los usuarios desactiven las reglas selectivamente durante determinados periodos de tiempo en los que no son necesarias. La instrucción *activar* permite que la regla vuelva a entrar en funcionamiento cuando tenga lugar el evento que la desencadena. La instrucción *eliminar* hace desaparecer a una regla del sistema.

Existe la posibilidad de agrupar las reglas en *conjuntos de reglas,* con un nombre, de tal manera que manera que todo el conjunto pueda ser activado, desactivado o eliminado. También es útil disponer de un comando que pueda activar una regla o un conjunto de ellas por medio de la sentencia PROCESAR REGLAS lanzada por el usuario.

2.3.5. Aplicaciones de las BBDD activas

Las aplicaciones del paradigma de bases de datos activas son muy variadas. Una primera clasificación de las aplicaciones lo establece el uso de éstas, internas o externas.

- **Internas** al SGBD: soportan las características propias del manejo o administración de la BD, tales como:

  - **Control de integridad**. Las restricciones que se pueden establecer para el control de integridad son:
    Restricciones estáticas: se evalúan sobre un estado de la base de datos.
    Restricciones dinámicas: se evalúan sobre la transición de un estado a otro.
    En función de quién especifica las restricciones se pueden dividir en:
    Restricciones "built-in": son fijas y especificadas con cláusulas del lenguaje de definición de datos. Por ejemplo: claves ajenas y claves primarias
    Genéricas: especificadas por el usuario, por ejemplo con la definición de *constraints* (Not null, unique, check).

  - **Mantenimiento de vistas y datos derivados**, los cuales pueden existir virtualmente o ser materializados.

  - **Mantener coherentes tablas replicadas:** de tal manera que siempre que se modifique la tabla maestra se actualice la réplica.

  - **Seguridad**

- **Externas**, las cuales son especificadas por el usuario y permiten realizan tareas específicas dependientes del dominio del problema. Estas aplicaciones contienen conocimiento de la aplicación, expresado en forma de reglas y reciben el nombre de *reglas de negocio.*

2.3.6. Ejemplo de diseño de disparadores

Se dispone de dos tablas, dentro de un a base de datos, en las cuales va a ser necesario utilizar reglas activas para mantener algunos campos derivados con valores correctos.

EMPLEADO

NOMBRE	NSS	SALARIO	ND	NSS_SUPERV

DEPARTAMENTO

NOMBRE	ND	SAL_TOTAL	NSS_JEFE

Cada empleado tiene un nombre, un número de seguridad social, un salario, un número de departamento (que es una clave ajena de departamentos y puede tener el valor nulo) y un supervisor directo (que es una clave ajena de la propia tabla de empleados).

Cada departamento tiene un nombre, un número, un salario total (campo derivado que es la suma de los salarios de todos los empleados de ese departamento) y un jefe (que es una clave ajena de empleados).

Para mantener el valor del atributo derivado SAL_TOTAL se puede emplear una regla activa. En primer lugar se deben determinar los eventos que pueden causar un cambio en el valor de este campo y estos son:

1. Insertar una ó más tuplas de empleados nuevos
2. Cambiar el salario de los empleados existentes
3. Cambiar la asignación de departamento de los empleados existentes.
4. Eliminar tuplas de empleados.

Para evento 1, sólo hace falta volver a calcular SAL_TOTAL si se asigna inmediatamente el nuevo empleado a un departamento, es decir si el valor de

ND de la tupla que se acaba de insertar no es nulo. Con lo cual, esta sería la condición a evaluar antes de llevar a cabo las acciones.

Para los eventos 2 y 4 podría evaluarse una condición similar, para verificar si el empleado cuyo salario ha cambiado ó ha sido eliminado, está asignado actualmente a algún departamento.

Para el evento 3 no se necesita ninguna condición, con lo cual será una regla EA (evento-acción). Sin embargo se necesitan llevar a cabo dos acciones: una para actualizar el SAL_TOTAL del antiguo departamento y otra para actualizar la del nuevo.

A continuación se muestran las 4 reglas activas especificadas en la notación SGBD Oracle:

```
R1:   CREATE TRIGGER SALTOTAL1
      AFTER INSERT ON EMPLEADO
      FOR EACH ROW
      WHEN (NEW.ND IS NOT NULL)
            UPDATE DEPARTAMENTO
            SET SAL_TOTAL = SALT_TOTAL + NEW.SALARIO
            WHERE ND= NEW.ND;

R2:   CREATE TRIGGER SALTOTAL2
      AFTER UPDATE OF SALARIO ON EMPLEADO
      FOR EACH ROW
      WHEN (NEW.ND IS NOT NULL)
            UPDATE DEPARTAMENTO
            SET SAL_TOTAL = SALT_TOTAL + NEW.SALARIO –
OLD.SALARIO
            WHERE ND= NEW.ND;

R3:   CREATE TRIGGER SALTOTAL3
      AFTER UPDATE OF ND ON EMPLEADO
```

```
        FOR EACH ROW
        BEGIN
                UPDATE DEPARTAMENTO
                SET SAL_TOTAL = SALT_TOTAL + NEW.SALARIO
                WHERE ND= NEW.ND;
                UPDATE DEPARTAMENTO
                SET SAL_TOTAL = SALT_TOTAL - OLD.SALARIO
                WHERE ND= OLD.ND;
                END;
R4:     CREATE TRIGGER SALTOTAL4
        AFTER DELETE ON EMPLEADO
        FOR EACH ROW
        WHEN (OLD.ND IS NOT NULL)
                UPDATE DEPARTAMENTO
                SET SAL_TOTAL = SALT_TOTAL - OLD.SALARIO
                WHERE ND= OLD.ND;
```

La cláusula ‘FOR EACH ROW’ empleada en las cuatro reglas indica que la regla se activa para cada tupla por separado. Esto se conoce con el nombre de *disparador de nivel de fila*. Si esta cláusula se omitiese el disparador se denominaría *disparador de nivel de sentencia* y sería activado una vez para cada sentencia disparadora.

3. OBJETIVOS

El propósito general del proyecto es el análisis, diseño e implementación de una aplicación relacionada con las Bases de Datos Activas, que sea capaz de:

Tomando una base de datos anteriormente diseñada en Microsoft Access, modelar el esquema de una base de datos relacional y toda su semántica, y poder conectarse a ella mediante una aplicación desarrollada en Caché.

Ésta aplicación permitirá crear una base de datos sin necesidad de disponer de conocimientos de Access.

Permitirá expresar toda la semántica de la BD, pudiendo dar de alta en la aplicación 'Check', 'Aserciones', 'Disparadores', los cuales son propios de un sistema de base da datos activas.

Finalmente, habrá una funcionalidad que permitirá, una vez dada de alta la estructura y semántica de la BD, generar en lenguaje SQL la creación de dicha base de datos. Por tanto, se creará un fichero de texto con el lenguaje de definición de datos apropiado para la creación de las tablas, aserciones y disparadores.

Para lograr todos estos objetivos de implementación se han tenido que tener buenas bases de conocimientos de los siguientes temas teóricos.

a) Conocimientos sólidos de bases de datos relacionales para la realización del diseño de la base de datos implementada en Microsoft Access.

b) Conocimientos de Diseño Orientado a Objetos de BD para realizar el modelo Entidad-Relación.

c) Conocimientos de bases de datos activas para saber como se introduce la semántica asociada a una BBDD.

d) Estudio del lenguaje de programación Caché para poder codificar el problema.

e) Estudio y conocimientos del lenguaje de definición de datos SQL.

4. ENTORNO DE TRABAJO

4.1. Introducción: qué es Caché

Caché (Intersystems) es una nueva generación de tecnología de bases de datos de alto rendimiento. Combina varios sistemas: una base de datos orientada a objetos, SQL de alto rendimiento y un potente acceso a datos multidimensionales. Desde cualquiera de ellos, se puede acceder simultáneamente a los mismos datos. Los datos sólo se describen una vez en un único diccionario de datos integrado, el cual está disponible instantáneamente utilizando todos los métodos de acceso gracias al concepto de arquitectura única que posee Caché.

Caché proporciona altos niveles de rendimiento, escalabilidad y una gran facilidad para una rápida programación de aplicaciones orientadas a objetos, que sería muy difícil desarrollar mediante tecnología relacional. Además, también permite utilizar un acceso a la base de datos de forma relacional, lo que hace a Caché muy versátil y adaptable a sistemas relacionales ya existentes. Este funcionamiento dual es lo que hace que se considere a Caché una base de datos Post-Relacional.

4.1.1. Arquitectura Única

La enorme potencia de Caché tiene su origen en su arquitectura única. En el núcleo del sistema se encuentra el motor de la base de datos de Caché, que proporciona un completo conjunto de servicios incluyendo: almacenamiento de datos, gestión de concurrencia, realización de transacciones y gestión de procesos. Se puede considerar el motor de Caché como un poderoso conjunto de herramientas para usar y gestionar la base de datos. Usando estas herramientas, Caché implementa un completo sistema de gestión de bases de datos orientada a objetos y relacional.

Los beneficios de esta arquitectura son múltiples:

- Los sistemas de bases de datos orientados a objetos y relacionales se comunican directamente con el motor de la base de datos, lo que supone operaciones mucho más eficientes; no existe ningún *middleware* objeto-relacional o tecnología puente SQL a objeto.

- Separación de la base de datos lógica de su implementación física, lo que hace posible la modificación substancial del desarrollo de una aplicación sin cambios en la lógica de la misma.

- Dado que la interfaz del motor de la base de datos es abierta, se puede hacer uso directamente de sus características donde se necesite. Esto posibilita desde la construcción de un sistema de gestión de la base de datos propio, hasta la posibilidad de añadir optimizaciones dirigidas a la realización de aplicaciones críticas.

- Posibilidad de hacer mejoras futuras sin causar un impacto en aplicaciones existentes.

4.1.2. Caché y las Bases de Datos Post-Relacionales

Caché está diseñada para proporcionar mejoras a las bases de datos relacionales ya existentes y superar sus limitaciones, así como soportar muchas de las herramientas basadas en SQL que hay en el mercado.

El término “relacional” de “post-relacional” se refiere al hecho de que Caché tiene todas funciones de una base de datos relacional. Los datos dentro de la base de datos de Caché están disponibles como si fuesen tablas de una verdadera base de datos relacional, y pueden ser consultados y modificados usando SQL estándar a través de ODBC, JDBC, o métodos de objetos.

El término “post” de “post-relacional” se refiere al hecho de que Caché ofrece una amplia gama de características que van más allá de los límites de las bases de datos relacionales, a la vez que sigue soportando la vista de los datos del estándar relacional. Estas características incluyen:

- La capacidad de modelar los datos como objetos, cada uno con una creación y sincronización automática a una representación relacional, mientras elimina los desajustes entre las bases de datos y los entornos de aplicación orientados a objetos, y reduciendo además la complejidad del modelado relacional.

- Un sencillo modelo de concurrencia basado en objetos.

- Tipos de datos definidos por el usuario.

- La capacidad para aprovechar las ventajas de los métodos, la herencia y el polimorfismo, dentro del motor de la base de datos.

- Extensiones de objetos para SQL para manejar identidades y relaciones de objetos.

- La capacidad para mezclar accesos SQL y orientado a objetos con una solicitud única, usando cada uno de ellos los métodos de accesos más adecuados.

- Control sobre la disposición física y la agrupación utilizada para almacenar los datos, con el fin de garantizar el máximo rendimiento en las aplicaciones.

Mientras muchas bases de datos con ambos accesos, relacional y orientado a objetos, proporcionan una forma de acceso por encima de la otra, en Caché ambos aspectos, relacional y orientado a objetos, van directamente a los datos. Esta dualidad de acceso directo es lo que da a Caché su poder post-relacional.

4.2. El Motor de la Base de Datos de Caché

En el centro de Caché se encuentra el motor de la base de datos. El motor de la base de datos está altamente optimizado para un mayor rendimiento, concurrencia, escalabilidad y fiabilidad. Existe un alto grado de optimización específica del sistema por plataforma para lograr el máximo rendimiento en cada una de las plataformas soportadas.

Caché es un sistema completo de bases de datos, esto incluye todas las características necesarias para ejecutar aplicaciones con misiones críticas, incluyendo transacciones, copias de seguridad y recuperación, y herramientas para administrar el sistema. Para ayudar a reducir el coste de operación, Caché esta diseñado para requerir significativamente menos administración que otras bases de datos.

Las principales características del motor de Caché se presentan a continuación.

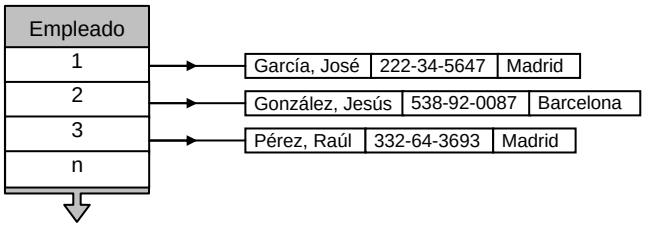
4.2.1. Almacenamiento Transaccional Multidimensional

Todos los datos dentro de Caché están guardados dentro de *sparse arrays* multidimensionales. A diferencia de las matrices multidimensionales utilizadas por los típicos productos OLAP (*OnLine Analytic Processing*, procesamiento analítico en línea), Caché soporta las operaciones de procesamiento de transacciones (inserciones, actualizaciones, bloqueos, transacciones) dentro de sus estructuras multidimensionales. También, a diferencia de la mayoría de los motores OLAP, estas estructuras multidimensionales no están limitadas al tamaño disponible de memoria. En lugar de esto, Caché incluye un sofisticado y eficiente sistema de caché de datos.

Dado que los datos de Caché son de por sí de longitud variable y se almacenan en *sparse arrays* (*arrays* en los que muchos valores de datos están repetidos o son nulos), Caché a menudo requiere menos de la mitad del espacio del necesario en una base de datos relacional. Además de reducir los requisitos de disco duro para almacenamiento de datos, mejora el rendimiento porque muchos más datos pueden ser leídos o escritos con una sola operación de E/S, y los datos pueden estar en caché de una forma más eficientemente.

4.2.1.1. Objetos y Almacenamiento Multidimensional

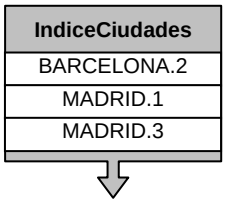
La tecnología orientada a objetos de Caché utiliza almacenamiento multidimensional como fundamento de su tecnología de persistencia de objetos. Por ejemplo, se supone que se tiene una clase simple denominada Empleado que representa los datos de los empleados de una empresa, con un nombre del empleado, un número de identificación y la ciudad donde vive. Las instancias de objetos de Empleado pueden ser almacenadas en una matriz multidimensional como la que se muestra aquí:



En este caso, el *array* es programado con un valor que identifica al objeto y los datos por cada instancia que son almacenados de forma compacta dentro de los nodos del *array*. Los objetos de Caché automáticamente crean la estructura de almacenaje óptima para las clases persistentes.

Si es necesario un índice, por ejemplo sobre la propiedad ciudad, el motor de persistencia podría usar un *array* multidimensional diferente, similar al del gráfico que se muestra más abajo, para asociar los valores con la ubicación

correspondiente a los identificadores de los objetos, normalmente se utiliza una estructura de dos dimensiones para este fin.



Estos índices se mantienen automáticamente cada vez que se introducen cambios en la base de datos. El motor SQL de Caché utilizaría automáticamente este índice para satisfacer las consultas para la búsqueda de empleados por ciudad.

4.2.1.2. Flexibilidad

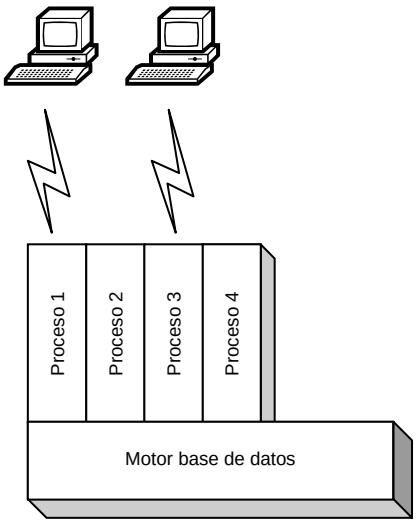
Los *arrays* multidimensionales dan a las aplicaciones un gran grado de flexibilidad gracias a la forma en la que se almacenan los datos. Por ejemplo, un conjunto de objetos estrechamente relacionados, como un objeto Factura y su correspondiente objeto LineaFactura (objeto que identificaría una línea simple de una factura), pueden ser fácilmente dispuestos para que los objetos LineaFactura sean agrupados físicamente con un objeto Factura para un acceso más eficiente.

Usando una característica conocida como "*subscript mapping*" (mapeo de subíndice), se puede especificar cómo los datos, que están dentro de uno o más *arrays*, son asignados a un archivo físico de la base de datos. Esta correspondencia es una tarea realizada por parte de administración de la base de datos y no requiere ningún cambio de la clase/tabla de definiciones o de la lógica de la aplicación. Por otra parte, el mapeo se puede hacer dentro de un *sparse array* específico, se puede mapear un rango de valores a una ubicación física, mientras se asigna otro valor a otro archivo, unidad de disco, o incluso a otro servidor de base de datos. Esto hace posible reconfigurar las aplicaciones Caché con poco esfuerzo.

La flexibilidad de las transacciones de almacenamiento multidimensional da a Caché una importante ventaja con respecto a las estructuras de dos dimensiones utilizadas tradicionalmente por las bases de datos relacionales: es esta flexibilidad la que permite a Caché tener más alto rendimiento SQL, orientado a objetos y de bases de datos XML. También significa que las aplicaciones Caché están mejor preparadas para futuros cambios en la tecnología.

4.2.2. Gestión de Procesos

Caché proporciona la capacidad para ejecutar operaciones de bases de datos, así como cualquier grado de lógica de negocio dentro de los procesos de Caché.



Un proceso es una instancia de una máquina virtual de Caché corriendo en un servidor Caché. Un servidor típico de Caché puede ejecutar miles de procesos simultáneos en función del hardware y del sistema operativo empleado. Cada proceso tiene acceso directo y eficiente al sistema de almacenamiento multidimensional.

La máquina virtual de Caché ejecuta instrucciones, que se denominan como "*P-code*" (código-P), que es un código muy optimizado para el uso de la base de datos, operaciones de E/S, y las operaciones lógicas normalmente requeridas para el procesamiento de transacciones y aplicaciones de almacenamiento de datos. El código de la máquina virtual puede ser creado de las siguientes maneras:

- SQL – Las consultas SQL enviadas a Caché son procesadas por el optimizador de *Caché SQL* que, a su vez, las convierte en *P-code* ejecutable y más eficiente (haciendo uso de cualquiera de los índices que puedan estar presentes).

- Comportamiento de Objetos – La tecnología de Objetos Caché proporciona un alto grado de funcionalidad orientada a objetos en el lado del servidor, como la persistencia de objetos, para la generación automática de *P-code* ejecutable. Usando generadores de métodos que hacen que se genere el código de los métodos de los objetos de acuerdo a unas normas predeterminadas.

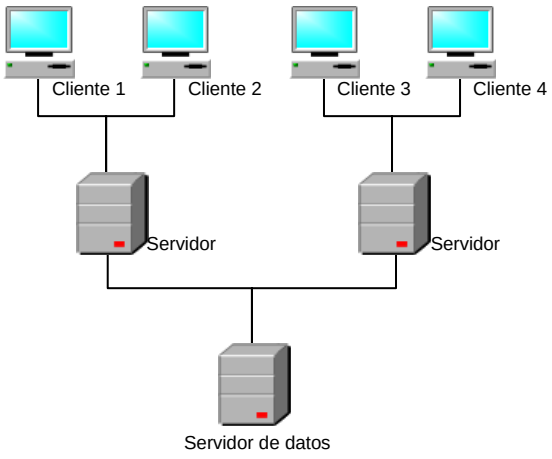
- *Caché ObjectScript* – Las aplicaciones pueden incluir cualquier lógica que se desee ejecutar dentro de los datos Caché o del servidor de aplicaciones usando el lenguaje de *script*, *Caché ObjectScript* [10]. Este código puede adoptar la forma de métodos de objetos, similar pero mucho más potentes que los procedimientos almacenados en el mundo relacional, ya que pueden hacer pleno uso de las características de orientación a objetos, o "rutinas" completas, que son pequeños programas que se ejecutan en un servidor Caché.

- *Caché Basic* – Los métodos de objetos también pueden ser implementados utilizando el lenguaje de programación *Basic* [8]. Caché incluye un poderosa, versión basada en objetos del popular lenguaje de programación Basic que es familiar a una gran parte del mundo los desarrolladores de software. *Caché Basic* se ejecuta en

cada plataforma soportada y es totalmente ínter-operable con *Caché ObjectScript*.

4.2.3. Gestión de Datos Distribuidos

Una de las principales características de Caché es su capacidad para poder unir servidores y formar una red de datos distribuidos. En esa red, las máquinas que sirven datos son conocidas principalmente como Servidores de Datos, mientras que las que realizan otras funciones de lógica del negocio, pero que no realizan ninguno o muy pocos procesos sobre datos, son conocidas como Servidores de Aplicaciones.



Los servidores pueden compartir los datos, y bloqueos, usando el Caché *Enterprise Cache Protocol* (ECP). ECP es eficaz ya que los datos son transportados en paquetes. Cuando se solicita la información a través de la red, el paquete de respuesta incluye los datos deseados, y datos adicionales que están relacionados con los primeros. Las relaciones naturales entre los datos son inherentes a los objetos y el modelo multidimensional de datos de Caché hace esto posible identificando e incluyendo esta información relacionada a la respuesta original. Esta información "asociada", es cacheada localmente en el cliente o en el servidor de aplicaciones. Por lo general, las solicitudes posteriores de datos se pueden satisfacer desde la caché local, evitando así peticiones adicionales a través de la red. Si el cliente realiza

cualquier cambio en los datos, sólo las actualizaciones realizadas se propagarán a la base de datos del servidor.

ECP hace posible que las aplicaciones soporten una amplia variedad de configuraciones en tiempo de ejecución incluyendo las multinivel y las *peer-to-peer*.

4.2.4. Gestión de Diario

Para proporcionar a la base de datos integridad y fiabilidad, Caché incluye una serie de subsistemas de diario que hacen un seguimiento de las actualizaciones físicas y lógicas de base de datos. La tecnología de gestión del diario también se utiliza para proporcionar soporte a las transacciones, por ejemplo, las entradas del diario son utilizadas para realizar las operaciones necesarias para hacer *rollback*, así como en el uso de base de datos espejo, el diario es utilizado para sincronizar el servidor espejo con el servidor principal de datos. Al igual que ocurre con el resto del sistema, Caché permite configurar el sistema de diario para satisfacer las necesidades de la aplicación a ejecutar.

4.2.5. Gestión de Bloqueos

Para apoyar los accesos concurrentes a las bases de datos, Caché incluye un potente sistema de gestión de bloqueos.

En los sistemas con miles de usuarios, reducir los conflictos entre los procesos es fundamental para proporcionar un alto rendimiento. Uno de los mayores problemas en las operaciones es aquel en el que varios usuarios desean tener acceso a los mismos datos. La gestión de bloqueos de Caché ofrece las siguientes características para aliviar este tipo de conflictos:

- Operaciones Atómicas – Para eliminar la típica situación de puntos calientes, Caché ofrece una serie de operaciones atómicas, es decir,

sin necesidad de bloqueos a nivel de aplicación. Un ejemplo de esto es la capacidad atómica de asignar valores únicos para la identificación del conjunto objeto/fila, punto en el que es común que aparezcan cuellos de botella en las aplicaciones relacionales.

- Bloqueos Lógicos – Caché no bloquea páginas enteras de datos mientras realiza las actualizaciones. Dado que la mayoría de las transacciones requieren acceso frecuente o cambios en pequeñas cantidades de datos, Caché ofrece bloqueos lógicos granulares que pueden ser tomados en base a un objeto (fila).

- Bloqueos Distribuidos – En las configuraciones de bases de datos distribuidas, Caché automáticamente proporciona bloqueos distribuidos.

4.2.6. Gestión de Dispositivos

Caché proporciona soporte para multitud de dispositivos, tales como archivos, TCP/IP, impresoras, lo que permite a las aplicaciones de Caché operar con otras tecnologías. Las opciones de conectividad disponibles con Caché, incluyendo CSP, ODBC, SOAP, y Java, se construyen en la parte superior del soporte subyacente.

4.2.7. Portabilidad

Caché se ejecuta y está optimizado, para una amplia variedad de plataformas hardware y sistemas operativos.

Se pueden adaptar fácilmente las aplicaciones desarrolladas con Caché, así como los datos de una plataforma a otra. Esta operación es tan fácil como instalar Caché en la nueva plataforma y mover los ficheros de la base de datos al nuevo sistema.

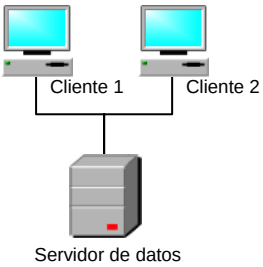
4.2.8. Opciones de Despliegue

Caché soporta una gran variedad de configuraciones de ejecución dándole mayor flexibilidad a la hora de implementar aplicaciones. Se puede cambiar entre las diferentes opciones de despliegue cambiando las configuraciones de sistema de Caché y sin que sea necesario normalmente cambiar la lógica de la aplicación.

Algunos casos típicos de despliegue se explican a continuación.

4.2.8.1. Configuración Básica Cliente/Servidor

La configuración más simple, cliente/servidor, consta de un único servidor de datos que sirve a varios clientes, que pueden ir desde uno a muchos miles, dependiendo de la aplicación y la plataforma.



Los sistemas clientes pueden ser cualquiera de los siguientes:

- Sistemas independientes ejecutando una aplicación cliente que se conecta a través de un protocolo cliente/servidor, como ODBC, ActiveX, JDBC, Java.

- A través de procesos de servidor Web comunicándose mediante CSP (*Caché Server Pages*) [11], SOAP [16], o alguna otra opción de

conectividad, como ODBC, JDBC. Cada servidor Web atenderá un número de sesiones ya sean peticiones de navegadores o de máquina a máquina.

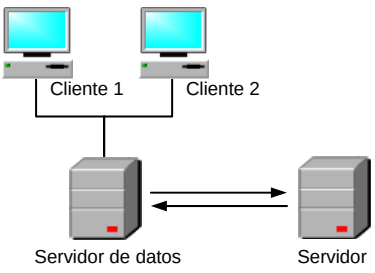
- Mediante procesos *Middleware*, por ejemplo, un servidor de aplicaciones *Enterprise Java Bean* que se conecta a Caché a través de ODBC, JDBC, etc.

- Dispositivos, tales como terminales o equipos de laboratorio, que se conecten a Caché utilizando uno de los muchos protocolos soportados, incluyendo telnet y TCP/IP.

- Alguna combinación de los anteriores.

4.2.8.2. Configuración Servidor Espejo/Sombra

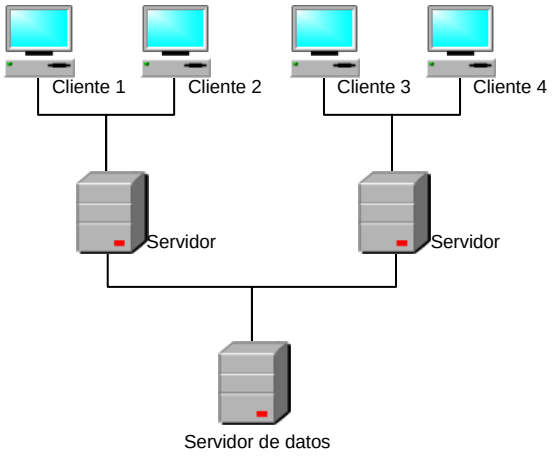
La configuración mediante un servidor espejo o sombra, se construye sobre la base de la configuración cliente/servidor añadiendo uno o más servidores espejo. Cada servidor espejo se sincroniza con los datos del servidor principal conectándose y monitorizando las transacciones del diario.



Los servidores espejos son normalmente utilizados para realizar consultas *ad hoc*, grandes informes y procesos por lotes, con el fin de limitar su impacto en el sistema de transacción principal. También se pueden usar para proporcionar un sistema de redundancia o respaldo para los casos de error.

4.2.8.3.Configuración Multinivel

La configuración multinivel usa la tecnología de base de datos distribuida de Caché, *Enterprise Cache Protocol* (ECP) para hacer posible un sistema que conecte un gran número de clientes.



En la configuración multinivel más simple, uno o más sistemas Caché, actúan como servidores de aplicaciones, son colocados entre el servidor central de datos y los diversos sistemas clientes existentes. En este caso, los servidores de aplicaciones no almacenan datos, en su lugar llevan a cabo los procesos que realizan el trabajo para el beneficio del cliente, quitando carga de CPU innecesaria al servidor de datos. Este tipo de configuración está mejor escalada para aplicaciones que presentan buena “localidad de referencia”, esto es, que la mayoría de las transacciones que se realizan tienen sus datos relacionados con el fin de que los bloqueos cruzados de los servidores de aplicaciones sean limitados. Estas aplicaciones, así como aquellas que realizan una gran cantidad de accesos de lectura, como la mayoría de las aplicaciones Web típicas, trabajan muy bien en este modelo.

También son posibles configuraciones más complejas, con múltiples servidores de datos, así como datos almacenados en servidores de aplicaciones.

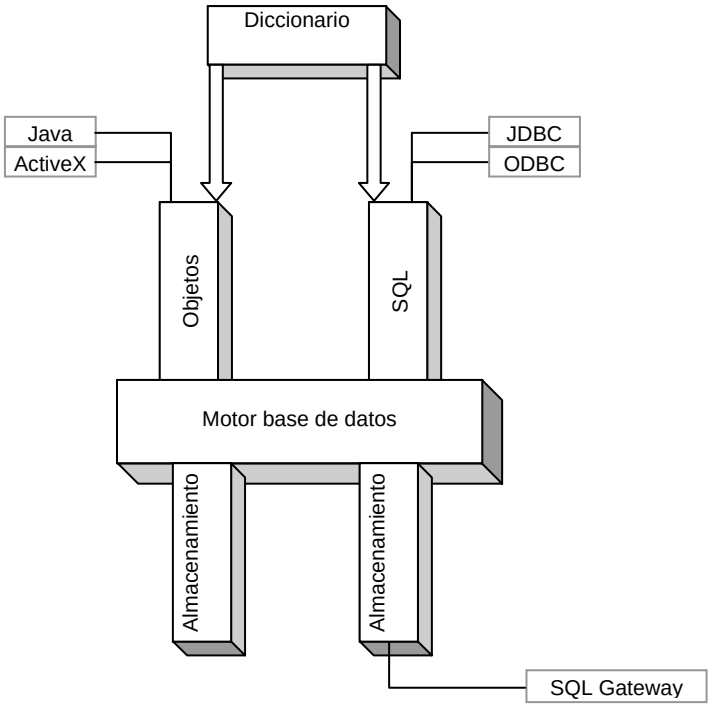
Normalmente las aplicaciones utilizan la configuración multinivel para una mayor capacidad de ampliación, así como para proporcionar alta disponibilidad, con servidores de aplicaciones funcionando como sistemas de reserva.

4.3. Objetos, SQL y Arquitectura Unificada de Datos

Una potente y singular característica de Caché es su Arquitectura Unificada de Datos, que proporciona alto rendimiento simultáneamente en el acceso a los datos almacenados en Caché, tanto en forma de objetos como de forma relacional.

4.3.1. Diccionario Unificado de Datos

Dentro de caché, se pueden modelar los componentes de la aplicación como objetos. Los objetos están organizados por clases que definen los datos de los objetos (propiedades) y su comportamiento (métodos).



La meta-información, o definición, de cada clase es guardada dentro de un repositorio común llamado diccionario de clases de Caché. El diccionario de clases es en si mismo un objeto de la base de datos, almacenado dentro de Caché, cuyos contenidos pueden ser accedidos usando objetos. El diccionario de clases, por medio de un compilador de clases, define la estructura de almacenamiento necesaria para la persistencia de los objetos y convierte las definiciones de clases en conjuntos paralelos de código ejecutable que proporcionan tanto acceso relacional como orientado a objetos a esta estructura de almacenamiento. Por medio de esta arquitectura, el código orientado a objetos y el código relacional son eficientes y se sincronizan automáticamente el uno con el otro.

Las definiciones de clases pueden ser añadidas al diccionario de clases de diferentes maneras:

- Interactivamente, utilizando el entorno de desarrollo *Caché Studio* [12].

- Relacionalmente, utilizando DDL (*Data Definition Language*). Caché acepta las declaraciones del estándar SQL DDL y crea automáticamente las clases correspondientes y las tablas de definiciones.

- Textualmente, utilizando XML. Caché soporta una representación de definición de clases externa con XML. Normalmente se utiliza para la gestión de código fuente, el despliegue, la generación automática de código, y la interoperabilidad con otras herramientas.

- Con programación, utilizando objetos. Usando el conjunto de definición de la clases de objetos de Caché, se pueden crear programas que se comuniquen directamente con el diccionario de clases y crear nuevas clases en la aplicación en tiempo de ejecución.

- Utilizando el asistente automático de Esquemas XML, incluido en *Caché Studio*, que puede crear definiciones de clases a partir de la mayoría de los archivos de esquemas XML.

4.3.2. Almacenamiento Flexible

El modelo de objetos de Caché se diferencia de los lenguajes de programación orientados a objetos en que además de añadir propiedades y métodos, se puede especificar el comportamiento relacionado con el almacenamiento, así como declarar índices, restricciones y definir su estructura de almacenamiento.

La estructura de almacenamiento utilizada para la persistencia de los objetos es independiente de la lógica de definición de una clase y es bastante flexible: los desarrolladores pueden utilizar las estructuras previstas por defecto por el compilador de clases o pueden modificarlas para casos específicos.

4.3.3. Modelo de Orientación a Objetos de Caché

Caché incluye un completo conjunto de características de bases de datos orientadas a objetos de última generación especialmente diseñada para satisfacer las completas necesidades, de las aplicaciones orientadas a transacciones.

El modelo orientado a objetos de Caché define el comportamiento y características de objetos persistentes, los almacenados en la base de datos y de objetos transitorios, aquellos que no se almacenan.

El modelo orientado a objetos de Caché incluye las siguientes características:

- Clases – Se pueden definir clases que representan el estado (datos) y el comportamiento (código) de los componentes de la aplicación. Las clases se utilizan para crear instancias de objetos como

componentes en tiempo de ejecución y como elementos almacenados dentro de la base de datos.

- Propiedades – Las clases pueden incluir propiedades, que especifican los datos asociados con cada instancia de objeto. Las propiedades pueden ser simples literales, como cadenas o enteros, de tipos definidos por el usuario, definidos mediante clases de tipo de datos, objetos complejos o incrustados, colecciones, o referencias a otros objetos.

- Relaciones – Las clases pueden definir cómo las instancias de objetos están relacionadas entre sí. El sistema proporciona automáticamente los métodos de navegación para las relaciones, así como integridad referencial en la base de datos.

- Métodos – Las clases pueden definir el comportamiento por medio de métodos: código ejecutable asociado con un objeto. Los métodos de los objetos se ejecutan dentro de un proceso del servidor de Caché, aunque pueden ser invocados desde un cliente remoto. Los métodos pueden utilizar secuencias de comandos de *Caché ObjectScript*, SQL, o pueden ser generados utilizando generadores de métodos, los cuales crean código de métodos personalizado automáticamente de acuerdo con las reglas definidas por el usuario.

- Persistencia de Objetos – La persistencia de los objetos es la capacidad de almacenar automáticamente y recuperar objetos de una base de datos. El sistema de persistencia incluye una completa funcionalidad de la base de datos, incluyendo la gestión automática de operaciones, el control de concurrencia, el mantenimiento de índices y la validación de datos. Los objetos persistentes son visibles automáticamente a través de consultas SQL.

- Herencia – Para derivar nuevas clases de las ya existentes, se puede reutilizar el código previamente escrito, así como crear versiones especializadas de estas clases.

- Polimorfismo – Caché soporta completamente el polimorfismo en los objetos. Esto significa que las aplicaciones pueden usar una interfaz conocida, como un conjunto de métodos y propiedades proporcionadas por una superclase, y el sistema automáticamente invocará la implementación de la interfaz de la aplicación correspondiente basándose en el tipo de cada objeto. Esto hace que sea mucho más fácil desarrollar aplicaciones de bases de datos flexibles.

- *Swizzling* (también conocido como "carga lenta") – Caché automáticamente trae a memoria desde disco cualquier objeto persistente relacionado con otros objetos cuando se hace referencia a ellos. Esto simplifica enormemente el trabajo con modelos de datos complejos.

La funcionalidad de orientación a objetos de Caché no es una parte separada de Caché; es la parte central de la programación dentro de Caché y está completamente integrada con el acceso relacional.

4.3.3.1. Clases y Objetos

Como se ha definido previamente una clase es una plantilla que describe los datos y el comportamiento de una entidad específica dentro de una aplicación. Por ejemplo, una aplicación puede definir una clase Coche que especifica las características compartidas por todos los coches dentro de esa aplicación.

Un objeto es una instancia específica de una clase. Por ejemplo, un "Seat Ibiza 2005" es una instancia específica de una clase Coche.

Los objetos pueden existir en la memoria de un determinado proceso o sesión donde se pueden manipular. Los objetos también pueden ser almacenados y recuperados de una base de datos.

Las clases proporcionan la estructura y la organización a las aplicaciones. En particular, las clases hacen posible dividir la ejecución de grandes aplicaciones entre desarrolladores o equipos de desarrolladores con la garantía razonable de que su labor más tarde pueda ser integrada.

Una característica importante de la Caché es que las nuevas clases se pueden definir en tiempo de ejecución, es decir, una aplicación puede extenderse mediante la creación y el uso de las nuevas clases generadas.

4.3.3.2. Referencia a un Objeto – OREF, OID y ID

Dentro de Caché, los objetos pueden encontrarse en disco o en memoria. Un objeto en disco es aquel que se ha almacenado en la base de datos; un objeto en memoria es aquel que se ha cargado desde la base de datos y puede manipularse. Se puede referenciar un objeto de diferentes maneras, dependiendo si está cargado en memoria o almacenado en la base de datos.

OREF

Una referencia a objeto. Un valor que especifica una referencia en memoria de una instancia de un objeto. Un OREF se refiere a una versión en memoria de un objeto. Cada vez que un objeto es traído a memoria, este puede tener un valor OREF diferente.

OID

Un identificador de objeto. Un valor que identifica de forma única a una instancia de un objeto almacenado en la base de datos. Un OID se refiere a una versión de un objeto en disco. Un OID identifica de forma única un objeto persistente que está almacenado en la base de datos. Una vez un objeto adquiere un valor OID, este valor no cambia.

ID

Un identificador de objeto. Un valor que identifica de forma única una instancia específica dentro de una extensión particular. Un ID referencia a una versión de un objeto en disco. Esta no incluye ninguna información específica de la clase.

Se puede usar un OID persistente de un objeto para localizar un objeto y traerlo a memoria; de forma similar, si se conoce un extensión de un objeto persistente, se puede usar este ID para localizarlo y traerlo a memoria. Una vez en memoria, el sistema le asigna un valor OREF que la aplicación podrá usar para referenciar al objeto y acceder a su contenido. Cuando un objeto persistente es almacenado en la base de datos, los valores de cualquiera de sus referencias de atributos esto es, las referencias de otros objetos persistentes, son almacenadas como valores OID. Para atributos de objetos que no tienen OIDs, el valor literal del objeto es almacenado con el resto del estado del objeto persistente.

4.3.3.3. Valores de OID e ID

Caché hace una distinción entre un identificador “universal” de objeto (OID) y un identificador “local” de objeto (ID). Un valor OID identifica de forma única cualquier objeto almacenado en la una base de datos. En concreto, un valor OID contiene un nombre de clase junto con un valor de ID. Un valor ID únicamente identifica un objeto perteneciente a una “extensión” específica. Una extensión es un conjunto de objetos relacionados, así como todas las instancias de una clase específica o todas las instancias de una clase y sus subclases.

Caché proporciona dos versiones de métodos para referenciar objetos persistentes: una usando valores OID y otra usando valores ID.

4.3.3.4. Tipos de Clases

Caché soporta una variedad de tipos de clases para cada comportamiento específico. Las clases están divididas en dos tipos principales, clases de tipos de datos y clases de objetos. Las clases de tipos de datos representan valores literales como cadenas de texto, enteros y fechas. Las clases de tipos de datos son usadas para crear propiedades literales de otros objetos. No tienen propiedades y no se pueden crear instancias de ellas.

Las clases de objetos pueden tener propiedades y pueden crear instancias. Muchas clases de objetos son subclases de una clase de sistema llamada %RegisteredObject, la cual proporciona automáticamente muchos de los comportamientos básicos de los objetos.

Una clase de objeto tiene un conjunto de funciones por defecto, las cuales incluyen:

- La creación de una instancia provoca la asignación automática de la memoria de sistema necesaria para sus propiedades.

- La creación de una instancia provoca la creación automática de un manejador de referencias (OREF) para la instancia.

- Soporta el polimorfismo.

Las clases de objetos se dividen de acuerdo a su comportamiento con la base de datos, en clases de objetos transitorios, persistentes y seriales.

4.3.3.4.1. Clases de Objetos Transitorios

Las instancias de las clases de objetos transitorios derivan directamente de la clase %RegisteredObject, son conocidos como objetos registrados o transitorios. Estos objetos tienen un completo conjunto de métodos

incorporados para controlar su comportamiento en memoria. No se almacenan en disco, sólo existen en memoria.

Una vez que se ha creado una instancia del objeto registrado, se puede referenciar mediante su OREF. Este valor asignado por el sistema es transitorio. El valor existe sólo mientras el objeto esté en memoria y no se garantiza que sea igual en diferentes invocaciones del objeto.

Los objetos registrados usan la memoria del sistema parar almacenar sus valores y soportan polimorfismo.

4.3.3.4.2. Clases de Objetos Persistentes

Las instancias de las clases persistentes derivan de la clase %Persistent y se pueden almacenar independientemente en la base de datos.

Al cargar un objeto persistente en memoria desde la base de datos, no se carga ningún otro objeto al que éste haga referencia. Tan pronto como el objeto sea referenciado, sin embargo, se trae a memoria automáticamente, esto es conocido como *swizzling*.

Cuando un objeto persistente es usado como propiedad, es decir, referenciado como una propiedad “referenciada”. Por ejemplo, si Doctor y Paciente son clases persistentes, se puede definir un atributo en la clase Paciente que usa Doctor como tipo:

```
Property ElDoctor As Doctor;
```

La propiedad, Eldoctor, es una propiedad referenciada. La primera vez que la propiedad es referenciada:

```
Set doc = paciente.ElDoctor;
```

El objeto Doctor apropiado es cargado desde la base de datos automáticamente. En este ejemplo, *doc* contendrá una OREF al objeto cargado Doctor.

Se pueden encadenar tantas referencias como se quieran dentro de una expresión:

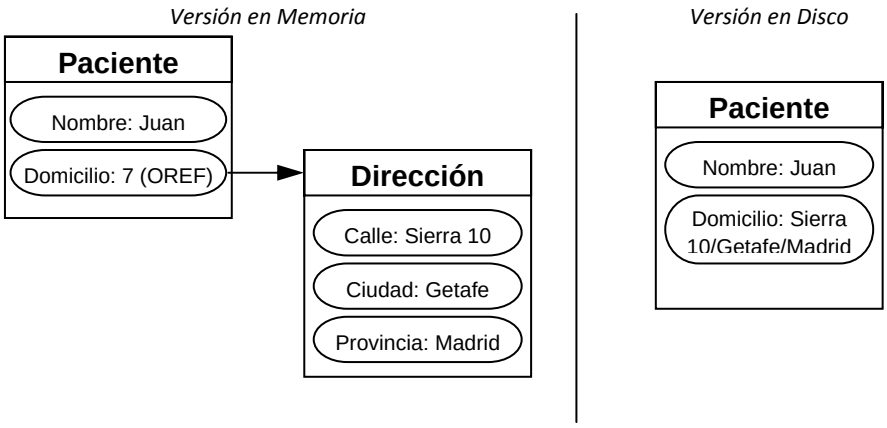
```
Set localizacion = paciente.ElDoctor.Hospital.Direccion.Ciudad;
```

Si cualquier referencia a objeto en una expresión no existe (es nula), entonces la expresión entera devolverá el valor nulo (“”).

4.3.3.4.3. Clases de Objetos Seriales

Las instancias de las clases de objetos seriales derivan de la clase %SerialObject y pueden ser embebidas dentro de otros objetos. Estos objetos pueden existir independientemente en memoria, pero cuando son almacenados en la base de datos, sólo pueden existir embebidos dentro de objetos persistentes.

En el siguiente diagrama se representa como un objeto embebido se encuentra en memoria y en disco:



1.3.3.4.4. Clases de Tipos de Datos

Las clases de tipos de datos definen y controlan valores literales. A diferencia de las clases de objetos, las clases de tipos de datos no tienen ninguna identidad independiente y nunca pueden estar explícitamente como instancias. En su lugar, existen sólo como propiedades de los objetos que los contienen. Las Clases de tipos de datos no pueden contener propiedades.

Un tipo de datos es una clase con la palabra clave ClassType que indica que es un “tipo de datos”. Las clases de tipos de datos pueden implementar un conjunto específico de métodos pertenecientes a la interfaz de tipos de datos. Esta interfaz incluye operaciones diseñadas para validar y para la interoperabilidad con SQL.

4.3.3.5. Definición de Clases

La forma más simple y común para definir clases en Caché es usar el entorno de desarrollo *Caché Studio* [12]. El estudio permite definir clases usando un formato simple de texto dentro de un editor de sintaxis coloreada o mediante el uso de una interfaz gráfica. Estas dos opciones son intercambiables y se sincronizan automáticamente.

A continuación se muestra una definición de la clase de un objeto persistente extremadamente simple, llamada Componente, tal y como se vería en *Caché Estudio*:

```
Class MyApp.Componente Extends %Persistent [ClassType = persistent]
{
Property Nombre As %String;
Property Valor As %Integer;
}
```

Esta clase está definida como una clase persistente, es decir, que puede almacenarse dentro de la base de datos. En este caso, Caché proporciona la clase %Persistent (los nombres de sistema comienzan con un carácter "%" para distinguirlos de la clase de la aplicación) que proporciona todo el código necesario para la persistencia a través de la herencia. La clase pertenece al paquete "MyApp". Los paquetes agrupan clases relacionadas y simplifican en gran medida el desarrollo de grandes aplicaciones. En la clase se definen dos propiedades: Nombre, que tiene un valor de cadena, y Valor, que tiene un valor entero.

Usando el código de *Caché ObjectScript* [10], por ejemplo dentro de un método, se puede utilizar esta sintaxis para manipular instancias de objetos de la clase Componente:

```
// Crear un nuevo objeto de la clase Componente
Set componente = ##class(MyApp.Componente).%New()
Set componente.Nombre = “Juan”
Set componente.Valor = 8

// Se guarda el nuevo objeto
Set sc = componente.%Save()
```

También usando *Basic* [8], se puede definir un método para manipular las instancias de objetos de la clase Componente:

```
‘ Crear un nuevo objeto de la clase Componente
componente = New Componente
componente.Nombre = “Juan”
componente.Valor = 8

‘ Se guarda el nuevo objeto en la base de datos
componente.%Save()
```

En este punto, una nueva instancia de Componente se almacena en la base de datos con un identificador de objeto único asignado por el sistema. Más tarde, se puede recuperar este objeto y abrirlo utilizando su identificador de objeto.

Esta recuperación se haría de la siguiente manera en *Caché ObjectScript*:

```
// Se abre una instancia
Set componente = ##class(MyApp.Componente).%OpenId(id)

// Se duplica el valor del objeto
Set componente.Valor = componente.Valor * 2

// Se guarda el objeto
Set sc = componente.%Save()
```

Y en *Basic* de la siguiente:

```
‘ Se abre una instancia
componente = OpenId Componente(id)

‘ Se duplica el valor del objeto
componente.Valor = componente.Valor * 2

‘ Se guarda el objeto
componente.%Save()
```

Se pueden realizar las mismas operaciones utilizando Java [15], C++ [13], u otros lenguajes vinculados con Caché. El compilador de la clase puede generar y sincronizar, cualquier código adicional necesario para acceder a objetos externos. Por ejemplo, si se está utilizando Caché con Java, se puede especificar al compilador de la clase que genere y mantenga automáticamente las clases Java que proporcionan acceso remoto a las clases persistentes de la base de datos de Caché. Dentro de un programa Java se puede utilizar este objeto, naturalmente:

```
// Se obtiene la instancia de Componente de la base de datos
componente = (MyApp.Componente)MyApp.Componente._open(db, new Id(id));

// Se escriben algunas propiedades del objeto
System.out.println("Nombre: ", + componente.getNombre());
System.out.println("Valor: ", + componente.getValor());
```

4.3.4. Caché SQL

Caché SQL [14] es un completo motor de base de datos relacional que está totalmente integrado con la tecnología Caché orientada a objetos. Además de las características estándar del SQL-92, Caché SQL ofrece:

- Soporta BLOBS (*Binary Large OBjects*, objetos binarios grandes) elementos utilizados para almacenar datos de gran tamaño que cambian de forma dinámica.

- Soporta procedimientos almacenados (implementados como métodos de objetos).

- Un conjunto de extensiones basadas en objetos.

- Tipos definibles por el usuario.

- Tiene soporte para índices transaccionales de mapa de bits.

Los índices de mapa de bits, por lo general utilizan un gran almacenamiento de datos y sistemas OLAP, ofrecen la posibilidad de realizar búsquedas de alta velocidad basadas en complejas combinaciones de condiciones. Estos índices

de mapa de bits no pueden ser actualizados en tiempo real, sin embargo suelen ser actualizados en procesos por lotes. Caché SQL soporta la combinación de índices de mapa de bits que ofrecen un alto rendimiento de búsqueda, al mismo tiempo que no se provoca ninguna pérdida en el rendimiento de inserciones y actualizaciones. Esto ofrece a las aplicaciones de procesamiento de transacciones la capacidad para llevar a cabo consultas sobre las bases de datos y al mismo tiempo llevan a cabo actualizaciones en tiempo real.

4.3.4.1. La Conexión Objeto/Relacional

Todos los componentes dentro del diccionario de Caché se definen como clases. El compilador de clases de Caché proyecta automáticamente las clases persistentes como tablas relacionales. Para cada característica de un objeto, hay una correspondencia equivalente en el modelo relacional, como se ilustra en el cuadro siguiente:

Características de Objetos	**Equivalente Relacional**
Paquete	Esquema
Clase	Tabla
Instancia de Objeto	Fila dentro de una tabla
Propiedad	Columna
Relación	Clave ajena
Objeto Embebido	Múltiples columnas
Método	Procedimiento
Índice	Índice

Cuando Caché carga declaraciones de SQL DDL (*Data Definition Language*, Lenguaje de Definición de Datos), se usa la inversa de esta proyección para crear clases que corresponden a las tablas relacionales.

Para demostrar la proyección relacional a objeto, se crea un ejemplo simple de estudio. Se realiza la definición de una clase persistente simple Persona, dentro del paquete “MyApp”, que contendrá dos propiedades, Nombre y Direccion:

```
Class MyApp.Persona Extends %Persistent [ClassType = persistent]
{
Property Nombre As %String(MAXLEN=100);
Property Direccion As Domicilio;
}
```

La clase Persona adquiere su comportamiento persistente de la superclase %Persistent proporcionada por Caché.

La propiedad Nombre esta definida como una cadena simple de texto de como máximo 100 caracteres.

La propiedad Direccion ilustra el uso de un tipo complejo definido por el usuario. En este caso una clase Domicilio que está definida como sigue:

```
Class MyApp.Domicilio Extends %SerialObject [ClassType = serial]
{
Property Ciudad As %String;
Property Provincia As %String;
}
```

La clase Domicilio deriva de la superclase %SerialObject. Esta clase proporciona la capacidad de poder ser serializada (convertida en una

representación de cadena simple de texto) e incrustarla en otra clase contenedora (como en este caso la clase Persona).

Cuando son vistos mediante SQL, la clase Persona tiene la siguiente estructura:

Tabla 1: Vista SQL de la clase Persona: SELECT * FROM Persona

ID	Nombre	Direccion_Ciudad	Direccion_Provincia
1	García, José	Madrid	Madrid
2	González, Jesús	Barcelona	Cataluña

El identificador de objeto ocupa una columna. Además, los campos del objeto embebido Domicilio se proyectan en campos separados. Estos campos reciben los nombres Direccion_Ciudad y Direccion_Provincia y su comportamiento es igual que si fuesen dos campos individuales.

4.3.4.2. Herencia y SQL

La herencia es una característica importante dentro del sistema basado en objetos pero no se da dentro de las bases de datos relacionales. Caché SQL hace posible usar la herencia usando construcciones relacionales estándar. Por ejemplo, se puede crear una nueva clase, Empleado, que deriva de la clase Persona usada en el ejemplo anterior.

```
Class MyApp.Empleado Extends Persona [ClassType = persistent]
{
Property Salario As %Integer(MINVAL=0,MAXVAL=10000);
}
```

Esta nueva clase extiende de la clase Persona y añade la propiedad adicional Salario.

Cuando se ve desde un punto de vista SQL, la clase Empleado tiene la siguiente estructura:

ID	Nombre	Direccion_Ciudad	Direccion_Provincia	Salario
3	López, Juan	Sevilla	Andalucía	22000

Todas las propiedades heredadas están disponibles como columnas. Únicamente se incluyen las filas que están instanciadas para Empleados. Si se hiciese a continuación una vista para la clase Persona las instancias serían las siguientes:

ID	Nombre	Direccion_Ciudad	Direccion_Provincia
1	García, José	Madrid	Madrid
2	González, Jesús	Barcelona	Cataluña
3	López, Juan	Sevilla	Andalucía

En este caso, se puede ver que en las filas devueltas en la consulta están todas las instancias porque cada Empleado es definido como una instancia de Persona. Sin embargo, en este caso, sólo se muestran las propiedades definidas para la clase Persona.

4.3.4.3. Extensiones de Objetos para SQL

Para hacer más fácil usar SQL dentro de aplicaciones orientadas a objetos, Caché incluye algunas extensiones para objetos que permiten usar SQL.

Una de las más interesantes extensiones es la capacidad de seguir las referencias de objetos usando el operador de referencia (“->”). Por ejemplo, si se tuviese una clase Vendedor que hiciera referencia a otras dos clases:

Contacto y Region. Se podrían referenciar las propiedades de las clases relacionadas usando el operador de referencia:

```
SELECT ID,Name,ContactoInfo->Name
FROM Vendedor
WHERE Vendedor->Region->Nombre = 'Antártica'
```

Por supuesto, también se puede expresar la misma búsqueda utilizando la sintaxis de SQL JOIN. La ventaja de la sintaxis del operador de referencia es que es breve y fácil de comprender de un vistazo.

4.4. Desarrollo de Interfaces

Caché proporciona herramientas para ayudar a desarrollar interfaces, así como, crear aplicaciones completas.

4.4.1. Caché Server Pages (CSP)

La primera da estas herramientas es la tecnología *Caché Server Pages* (CSP) [11]. CSP es a la vez una arquitectura y un conjunto de herramientas usadas para crear aplicaciones interactivas CSP. Con CSP, se pueden construir y desarrollar aplicaciones Web de alto rendimiento y escalabilidad. CSP permite generar páginas Web dinámicas, normalmente usando datos de la base de datos de Caché. Estas páginas son dinámicas, esto es, que la misma página puede mostrar diferente contenido cada vez que son pedidos los recursos de datos dinámicos.

CSP es versátil. Puede hacer lo siguiente:

- Mostrar los datos de inventarios que cambian minuto a minuto.

- Soportar comunicaciones Web con miles de usuarios activos.

- Personalizar páginas basadas en la información de usuario almacenada en la base de datos Caché.

- Personalizar páginas basadas en los datos de usuario para distintos usuarios, en función de sus necesidades y sus permisos de seguridad.

- Servir HTML, XML, imágenes, u otro tipo binario archivos o de datos textuales.

- Rápido rendimiento en la entrega del contenido, porque está estrechamente ligado al alto rendimiento de la base de datos de Caché.

Además de proporcionar un acceso rápido a la base de datos de Caché, proporciona una serie de características esenciales para el desarrollo de aplicaciones de bases de datos basadas en Web, incluyendo la gestión de sesiones, autenticación, y la capacidad para llevar a cabo operaciones interactivas en la base de datos desde una página Web.

4.4.2. Zen

El segundo instrumento complementario es Zen [17]. El marco de desarrollo de aplicaciones Zen proporciona una forma sencilla de crear rápidamente complejas aplicaciones Web ricas en datos mediante el ensamblado de componentes previamente construidos. Estos componentes crean automáticamente el HTML estándar y el código JavaScript necesario para hacer las aplicaciones Web. Además, proporcionan un modelo de objetos

común que es compartido entre el navegador del usuario y la lógica de la aplicación que se ejecuta en el servidor.

Zen se basa en *Caché Server Page* (CSP) y en la tecnología de Base de Datos Orientada a Objetos de Caché. Esta tecnología ofrece una plataforma robusta, escalable y portátil para almacenar aplicaciones Web.

4.5. Conectividad

Caché incluye una variedad de tecnologías para una eficiente y fácil conexión e interoperabilidad con casi todas las demás arquitecturas de software.

4.5.1. Interfaces de llamada de Caché

Caché proporciona una interfaz de llamadas entrantes (*callin*) que pueden utilizarse dentro de programas en C para ejecutar comandos y evaluar expresiones de Caché. La interfaz de llamadas entrantes permite una amplia variedad de aplicaciones. Por ejemplo, se puede utilizar para tener disponible *Caché ObjectScript* en un menú integrado o GUI. Si se quiere obtener información de un dispositivo externo, como un cajero automático o un elemento de un equipo de laboratorio, la interfaz de llamadas entrantes permite almacenar estos datos en una base de datos Caché. Aunque Caché actualmente sólo soporta programas en C y C++, cualquier lenguaje que utilice la llamada estándar para esta plataforma (OpenVMS, UNIX, Windows) puede invocar las funciones de llamada entrantes.

Caché también provee una interfaz de llamada saliente (*callout*), que es un conjunto de funciones y utilidades que permiten acceder a comandos externos desde dentro de Caché. Con la interfaz de llamada saliente, se puede:

- Llamar a un número de comandos de los sistemas operativos de varias plataformas.

- Llamar a rutinas escritas en otros lenguajes.

- Construir sus propias llamadas a rutinas, como una DLL (en Windows), una biblioteca compartida (en UNIX), o compartir un ejecutable (en OpenVMS), en lugar de vincularlos estáticamente.

- Referenciar sus propios módulos de llamadas, que pueden ser personalizados mediante bibliotecas de vínculos dinámicos (Windows), bibliotecas dinámicas compartidas (UNIX), o imágenes ejecutables compartidas (OpenVMS).

4.5.2. Caché ODBC

El interfaz a nivel de llamadas del lenguaje C para Caché SQL es ODBC. A diferencia de otros productos de bases de datos, el controlador ODBC de Caché es un controlador nativo, que no está construido con ninguna otra interfaz de propietario. El controlador ODBC de Caché ofrece las siguientes características:

- Alto rendimiento

- Portabilidad

- Soporte Unicode nativo

- Hilo de seguridad

Se puede utilizar Caché ODBC con cualquier herramienta, aplicación o entorno de desarrollo que soporte ODBC.

4.5.3. Caché JDBC

Caché incluye un controlador de muy alto rendimiento, nativo de JDBC para facilitar el acceso a las aplicaciones y herramientas Java relacionales. Al igual que el controlador ODBC de Caché, el controlador JDBC ofrece un alto rendimiento, portabilidad, soporte Unicode nativo, e hilo de seguridad. El controlador se puede utilizar con cualquier herramienta, aplicación o entorno de desarrollo que soporte JDBC.

4.5.4. Caché XML y Servicios Web

Caché incluye un múltiples formas de interactuar con XML y SOAP [16].

En primer lugar, se incorpora soporte para proyecciones XML de cualquier objeto, así como de conjuntos de datos. Caché trae el poder de los objetos al procesamiento de XML: no está limitado al uso de XML como archivos de texto simple o relacional BLOBS, ni se está obligado a introducir documentos XML complejos dentro de filas y columnas relacionales. En lugar de ello, se pueden utilizar objetos como representaciones directas de documentos XML y viceversa. Porque Caché incluye una base de datos orientada a objetos nativa, que puede utilizar esos objetos directamente con la base de datos; no se necesita complejos *middleware* o tecnologías de conversión. Se puede utilizar XML con Caché de varias de maneras, incluyendo:

- Como un formato estándar en aplicaciones de mensajería. Esto incluye protocolos estándar de la industria, así como otras soluciones creadas por particulares.

- Como un formato estándar para el intercambio de datos entre aplicaciones y usuarios.

- Como un estándar para la representación externa de almacenamiento de datos. Esto puede incluir registros de bases de datos tradicionales o puede incluir contenidos más complejos como documentación.

Se puede importar documentos XML en objetos Caché, manipular objetos XML en Caché, y exportar objetos Caché como documentos XML. Se puede también tener el control sobre los esquemas XML utilizados por estos objetos, y se puede exportar también el esquema.

El soporte de Caché para XML hace que sea fácil para soportar también SOAP y Servicios Web. El soporte Caché para SOAP es fácil de usar, eficaz y plenamente compatible con la especificación SOAP. Este soporte se basa en Caché y no requiere ningún complejo *middleware* o extensión del sistema operativo. Está disponible en cada plataforma soportada por Caché. Utilizando Caché SOAP, se puede hacer lo siguiente:

- Definir y publicar Servicios Web, mediante una colección de métodos relacionados que las aplicaciones cliente pueden invocar utilizando el protocolo SOAP. Estos métodos pueden ser descubiertos e invocados por otras aplicaciones SOAP. Caché ejecuta los métodos SOAP directamente dentro de la base de datos, con lo que la ejecución de tales métodos es muy eficiente.

- Proporciona un medio para que las aplicaciones puedan interactuar con otras aplicaciones utilizando el protocolo SOAP.

- Desarrollar Servicios Web altamente eficiente, basados en nuevos desarrollos o quizás utilizando el código de aplicaciones existentes, que se pueden desplegar en cualquier plataforma soportada por Caché no limitándose a un determinado sistema operativo o plataforma.

4.5.5. Servidor de Objetos Caché para Java y EJB

La adaptación de Caché para Java permite utilizar las clases persistentes de Caché con aplicaciones Java y EJB (*Enterprise Java Bean*).

En primer lugar, la adaptación de Caché para Java proporciona una forma simple y directa de utilizar Objetos de Caché dentro de una aplicación Java. Se pueden crear aplicaciones Java que funcionen con la base de datos Caché de las siguientes maneras:

- Adaptación Caché para Java – La adaptación Caché para Java permite a las aplicaciones Java trabajar directamente con los objetos del servidor de Caché. La adaptación crea automáticamente las clases Proxy Java para cada especificación de clases de Caché. Cada clase Proxy es una clase Java pura, que sólo contiene código Java estándar y que proporciona a la aplicación Java acceso a las propiedades y los métodos de la clase Caché correspondiente.

- Adaptación Caché para Java ofrece soporte completo para la persistencia de objetos en la base de datos, incluyendo control de concurrencia y de transacciones. Además, existe un sofisticado sistema de almacenamiento caché de datos para reducir al mínimo el tráfico de red cuando el servidor Caché y el entorno Java se encuentran en máquinas separadas.

- El controlador JDBC de Caché incluye un controlador JDBC de nivel 4 (Java puro) que soporta la versión 3.0 del API JDBC. El controlador JDBC de Caché proporciona un alto rendimiento de acceso relacional a Caché. Para conseguir la máxima flexibilidad, las aplicaciones pueden utilizar JDBC y la adaptación de Caché para Java al mismo tiempo.

En segundo lugar, la adaptación Caché para EJB permite usar Caché con aplicaciones *Enterprise Java Bean* (EJB). La adaptación Caché EJB ofrece las siguientes ventajas:

- Adaptación Caché EJB proporciona el mismo rendimiento del código persistente escrito a mano sin el esfuerzo de escribirlo y mantenerlo. Caché genera automáticamente código de persistencia para Entidades EJB utilizando los meta-datos almacenados dentro del diccionario de clases de Caché.

- La base de datos de Caché se define en términos de objetos, por lo que no hay necesidad de engorrosos mapeos objeto-relacional en el servidor EJB.

- Caché EJB ofrece un sofisticado almacenamiento en memoria caché para eliminar el tráfico de red innecesario entre el servidor de EJB y el servidor de base de datos.

- Caché permite el desarrollo más rápido de aplicaciones EJB mediante la eliminación de gran parte del trabajo complejo. Además de la generación automática de Entidades EJB, Caché genera descriptores de despliegue EJB, *scripts* para el despliegue de su entidades en el servidor de EJB, y código Java para pruebas de Entidades.

- Caché permite a las aplicaciones EJB usar tanto acceso orientado a objetos como relacional a la base de datos, permitiendo que se elija la técnica más apropiada para cada tarea.

4.5.6. Servidor de Objetos Caché para ActiveX

El servidor de objetos Caché para ActiveX pone a disposición las clases existentes en Caché para que sean usadas dentro de entornos ActiveX, como Visual Basic o .NET. Esto incluye los siguientes componentes ActiveX:

- Servidor de Objetos Caché para ActiveX – Un servidor de automatización de ActiveX que presenta objetos de Caché como objetos ActiveX.

- Controlador de Listas de Caché – Un controlador ActiveX escrito para Visual Basic que ayuda a la visualización de los resultados de las consultas. Se debe proporcionar la interfaz para seleccionar las consultas y los parámetros de consulta para su ejecución.

- Controlador de Consultas de Caché – Un controlador ActiveX escrito para Visual Basic que proporciona una interfaz simple para la ejecución de consultas y la muestra de los resultados. El Controlador de Consultas de Caché proporciona una interfaz para seleccionar en tiempo de ejecución cualquier consulta que devuelva el identificador (ID) y para especificar cualquier parámetro de la consulta.

- Asistente para Objetos Caché – Un complemento a Visual Basic que le permite crear rápida y fácilmente formularios sencillos para acceder a las propiedades de una clase de Caché.

La adaptación ActiveX de Caché proporciona acceso a Caché desde cualquier aplicación que soporte objetos ActiveX. Este sistema permite que se puedan crear instancias y manipular objetos Caché con Visual Basic y otras aplicaciones cliente, y proporciona una interfaz transparente para crear instancias de objetos en el lado del servidor. Este sistema proporciona un

conjunto especial de herramientas sólo par Visual Basic, que se denomina *Visual Caché*.

4.5.7. Servidor de Objetos para C++

La adaptación de Caché para C++ hace que las clases Caché estén disponibles para su uso en aplicaciones C++.

- La Adaptación de Caché para C++ – esta adaptación permite trabajar a aplicaciones C++ con objetos del servidor de Caché. El generador de clases Caché puede crear una clase Proxy en C++ para cualquier clase de Caché. Las clases Proxy contienen código C++ estándar que puede ser compilado y utilizado en aplicaciones C++, facilitando el acceso a las propiedades y los métodos de la clase correspondientes en Caché.

La adaptación de C++ ofrece un completo soporte para la persistencia de objetos en la base de datos, incluyendo el control de la concurrencia y de las transacciones. Además, existe un sofisticado sistema de almacenamiento de datos en memoria caché para reducir al mínimo el tráfico de la red cuando el servidor Caché y las aplicaciones en C++ se encuentran en diferentes máquinas.

- Adaptación Dinámica – En lugar de clases Proxy compiladas en C++, se puede trabajar con clases Caché dinámicamente, en tiempo de ejecución. Esto puede ser útil para escribir aplicaciones o emplear herramientas que se ocupan de clases generales y no dependan de clases particulares de Caché.

- Adaptación ligera – esta adaptación es un subconjunto limitado de la biblioteca de Caché C++ destinada principalmente a la carga simple de datos a muy alta velocidad. Combina la aplicación en C++ y el Servidor de Objetos de Caché en un solo proceso, utilizando

comunicaciones internas entre procesos en lugar de TCP/IP para intercambiar datos entre ellos. Para la manipulación básica de objetos (creación, apertura por identificador, actualización y eliminación de objetos), es de diez a veinte veces más rápido que el adaptador estándar de C++.

- El controlador ODBC de Caché – El adaptador C++ ofrece clases especiales para encapsular la complejidad de ODBC. Para conseguir la máxima flexibilidad, se puede utilizar aplicaciones ODBC y el adaptador de Caché C++ al mismo tiempo.

4.5.8. Gateway de Caché SQL

El *Gateway* de Caché SQL da acceso a los objetos de aplicaciones Caché a otras bases de datos relacionales de terceras partes. Usando el *Gateway* SQL, las aplicaciones pueden:

- Acceder a los datos almacenados en bases de datos de terceros con aplicaciones que utilizan Objetos Caché y/o consultas SQL.

- Almacenar objetos persistentes de Caché en bases de datos relacionales externas.

4.5.9. Gateway Caché ActiveX

El *Gateway* ActiveX da acceso directo a aplicaciones Caché a componentes ActiveX/COM/.NET. Por medio de clases envoltorio, los componentes ActiveX están disponibles como instancias de clases de objetos Caché y pueden ser utilizados de la misma manera que cualquier otra clase. *Caché Activate* proporciona la capacidad para crear una instancia externa de un objeto COM y manipularla como si fuera un objeto nativo Caché.

Caché Activate funciona de la siguiente manera:

1. Usando el asistente de *Caché Activate*, se pueden crear una o más clases de envoltura. Estas clases Caché proporcionan métodos que corresponden a la interfaz de un componente ActiveX.

2. Dentro de una aplicación Caché, se puede crear una instancia de una clase envoltorio de ActiveX. *Caché Activate* de forma transparentemente crea una instancia del componente adecuado de ActiveX dentro del mismo proceso. Al invocar los métodos de la clase, automáticamente se envían al método adecuado de la interfaz ActiveX.

4.6. Herramientas de desarrollo y Útiles para la Base de Datos

Caché incluye una serie de herramientas de desarrollo de aplicaciones, así como utilidades para la administración de la base de datos. En un sistema Windows, estas son accesibles desde el icono de cubo  de Caché dentro de la barra de tareas de Windows. Para ver las herramientas disponibles se debe hacer clic derecho sobre el icono. Si no ve este icono, se debe abrir el menú Inicio de Windows, buscar la carpeta de Caché dentro de Programas y hacer clic en el programa Caché.

Las diversas utilidades gráficas son aplicaciones cliente/servidor cuyos clientes se ejecutan en sistemas Windows, pero pueden ser usadas de forma remota desde Windows, Linux, UNIX y OpenVMS.

Las herramientas de desarrollo de Caché y utilidades incluyen:

4.6.1. Caché Studio

Caché Studio [12] es el entorno base de desarrollo para Caché. El estudio es un entorno visual de desarrollo que permite crear bases de datos orientadas a objetos y aplicaciones con Caché. Entre las principales características que ofrece *Caché Studio* están:

- Un editor integrado en un único entorno que permite crear clases, incluyendo clases persistentes, páginas CSP (Caché Server Pages), y rutinas de *Caché ObjectScript*.

- Sistema de coloreado y control de sintaxis integrado para el control de *Caché ObjectScript*, *Basic*, *Java*, SQL, *JavaScript*, HTML y XML.

- Soporte para el desarrollo de aplicaciones por parte de equipos de desarrolladores que trabajan con un repositorio común de código fuente.

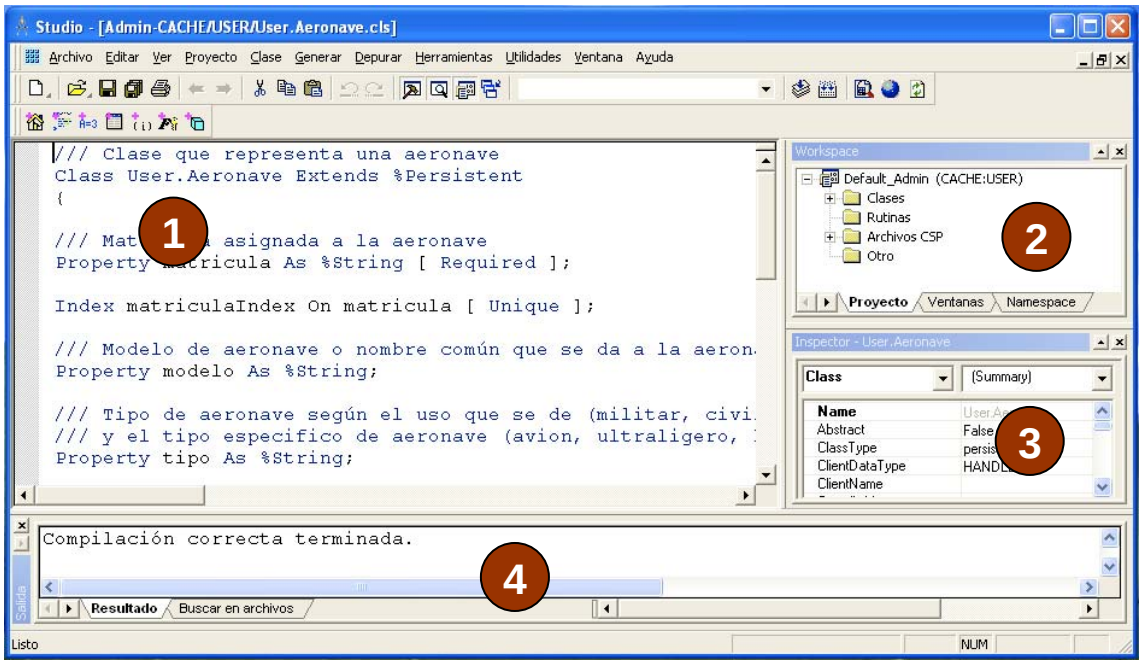
- Depurador de código fuente gráfico.

- La capacidad de organizar el código fuente de la aplicación en proyectos.

Caché Studio es una aplicación cliente, construida con Objetos Caché, que se ejecuta en Sistemas Operativos Windows. Puede conectarse a cualquier servidor Caché compatible con la versión actual de Caché Studio independientemente de la plataforma y sistema operativo del servidor donde está instalado Caché.

4.6.1.1. Visión General

Caché Studio es una aplicación estándar de Windows. Utiliza ventanas para mostrar y permitir la edición de los diversos elementos de la aplicación. Los principales componentes de la interfaz de usuario de *Caché Studio* se muestran a continuación:



1. Ventana de Edición, puede servir para:
    a. Editor de Clases: para la edición de definiciones de clase.
    b. Editor de rutinas: para la edición de rutinas.
    c. Editor de CSP: para la edición textual de CSP.
2. Ventana de Espacios de Trabajo: contiene tres pestañas que permiten visualizar:
    a. El contenido del proyecto actual.
    b. Todas las ventanas abiertas actualmente.
    c. El contenido del espacio de nombres actual.
3. Ventana del Inspector de clases: permite ver y modificar las palabras clave en una definición de clase.
4. Ventana de Salida: muestra las salidas del Servidor Caché, como por ejemplo los mensajes generados durante compilación de una clase.

Además de las ventanas mostradas arriba, Caché Studio contiene asistentes para ayudar con las tareas más comunes. Estos asistentes son:

- Asistente de Nueva Clase: define una nueva clase.
- Asistente de Nueva Propiedad: añade una nueva propiedad a una definición de clase.
- Asistente de Nuevo Método: añade un nuevo método para una definición de clase.
- Asistente de Relación: añade o modifica una relación entre las múltiples definiciones de clase.
- Asistente de Formularios Web: crea un formulario HTML vinculado a un objeto Caché en una página CSP.
- Asistente *Caché Active*: crea una clase envoltorio *Caché Active* para componentes ActiveX.

4.6.2. Portal de Gestión del Sistema Caché

El Portal de Gestión del Sistema proporciona una interfaz basada en Web para la gestión de Caché. El portal de gestión incluye herramientas para los administradores de sistemas, la gestión de la seguridad, operadores de base de datos, y para cualquier otra necesidad de acceso a Caché. Sus características administrativas le permiten configurar un sistema Caché, ver y modificar sus parámetros de configuración, ajustar la configuración del sistema, y crear y editar bases de datos y espacios de nombres. Las características del navegador Web de la base de datos incluye la posibilidad de examinar el contenido de la base de datos de Caché, navegar por las clases y rutinas

existentes, así como supervisar la actividad de la base de datos y realizar operaciones de copias de seguridad. Sus características relacionadas con la seguridad le permiten añadir, modificar y eliminar usuarios, roles, recursos, privilegios, y llevar a cabo otras tareas de gestión de la seguridad. Sus características relacionadas con SQL proporcionan una vista gráfica, basada en SQL de la base de datos de Caché; se puede usar para administrar los roles y permisos SQL, navegar por las tablas y ver sus definiciones, ejecutar consultas SQL *ad hoc*, gestionar la caché de consultas, e importar y exportar datos.

4.6.3. Caché Terminal

Caché Terminal [2] proporciona una interfaz interactiva, de línea de comandos para Caché. Se puede utilizar el Terminal para interacciones directas contra el motor de base de datos Caché. El Terminal es una herramienta importante para realizar las pruebas y solucionar los problemas de las aplicaciones.

Dentro de la terminal de Caché, se trabaja dentro de un único espacio de nombres. El *prompt* que aparece en el Terminal de Caché indica el espacio de nombres en el que se está trabajando actualmente. Por ejemplo:

```
USER>
```

El Terminal Caché se utiliza principalmente para las siguientes tareas:

- Introducción de comandos de Caché de todo tipo. Por ejemplo:

```
USER> Set nombreDirectorio = “C:\test”
```

```
USER> Set obj = ##class(TestMyClass).%New()

USER> Write obj.Prop1
```

- Ejecución de secuencias de comandos de Terminal (*script*), que son archivos con la extensión *.scr*, existentes en el sistema de archivos. La Terminal dispone de una pequeña serie de comandos que se pueden usar en estos *scripts*, incluyendo un comando que envía un Comando Caché a la Terminal, como si se hubiera escrito de forma manual.

A continuación se muestra una vista general del Terminal:

5. DESARROLLO DE APLICACIONES EN CACHÉ

5.1 El motor de datos multidimensional

A diferencia de las bases de datos interdependientes, que fuerzan los datos en tablas de dos dimensiones, Caché almacena datos en formaciones multidimensionales. Junto con permitir un modelado realista de los datos, las formaciones multidimensionales son mucho más rápidas de acceder, al eliminar el proceso superpuesto asociado con “tabla de salto” y “uniones” que simbolizan la tecnología interdependiente. Otra característica de gestión mejorada es el Protocolo Distribuido único de Caché, que reduce dramáticamente el tráfico en redes de trabajo de sistemas distribuidos. En pruebas a clientes, Caché ha funcionado hasta 20 veces más rápido que las bases de datos interdependientes.

Aunque los datos están almacenados en forma multidimensional, Caché les da a los desarrolladores la libertad para modelarlos de la forma que elijan: como objetos, tablas, ó como formaciones multidimensionales. Caché viene con una interface gráfica fácil de usar para modelar Objetos Caché. También acepta entrada desde Rational Rose (una herramienta para modelar objetos) y archivos DDL (el estándar para definir tablas interdependientes).

En virtud de la Arquitectura de Datos Unificados de Caché, todos los datos son accesibles automáticamente como objetos y tablas. Nunca hay necesidad de “mapear” desde una forma a la otra, y no se necesita ningún proceso superpuesto para convertir entre formas. La Arquitectura de Datos Unificados aumenta la gestión y la productividad.

Caché también permite elegir cuando se trata de bases de datos y escritura lógica de negocios. La Escritura de Objetos Caché soporta todos los métodos de acceso a datos: objetos, SQL, multidimensional e incluso HTML. Caché

Básico es similar a Visual Basic, sólo se necesitan pocas modificaciones para tomar ventaja de las capacidades únicas de Caché.

5.2 Acceso a la Web

En concordancia con los valores esenciales de InterSystems, la conectividad a Web de Caché está orientada hacia la entrega de una graduación masiva y de alta gestión, unida a una plataforma de desarrollo de aplicaciones súper-rápida.

En la arquitectura de Web única de Caché, las Páginas del Servidor Caché se ejecutan en el servidor de datos, junto a los datos a los que se necesita acceder. No sólo hace de esta aproximación una acelerada gestión, sino también aumenta enormemente la graduación, realizando parte importante del proceso de carga fuera del servidor de Web, dejándolo libre para manejar más requerimientos del buscador de consultas.

Caché aplica la rápida fuerza de desarrollo de la tecnología de objetos a la creación de Páginas del Servidor Caché. Cada Página del Servidor Caché es un objeto por sí sola, y puede heredar el comportamiento del administrador de sesión (de varios niveles de seguridad) desde el sistema de objetos proporcionado por InterSystems. Esto libera a los desarrolladores de aplicación de mucha de la tediosa codificación del sistema-nivel necesaria para mantener "el estado" durante las sesiones de usuario. La herencia de objeto también es una forma rápida de asegurar una “mirada” consistente a través de todas las páginas de una aplicación.

Adicionalmente, Caché simplifica el desarrollo en Web permitiendo a los diseñadores de Web y desarrolladores de aplicación trabajar en paralelo para lograr el resultado final. Usando herramientas familiares autorizadas para Web, siempre disponibles, los diseñadores de Web agregan funcionalidad a las páginas al incorporar Rótulos de Aplicación de Caché (RAC), de la misma manera que ellos harían cualquier rótulo HTML estándar. Los RACs para

algunas funciones estándar vienen con Caché, o se pueden construir bajo especificaciones individuales. Los desarrolladores de aplicación pueden escribir RACs que ejecuten funciones útiles, independientemente del diseño de la página que los contiene. Como resultado, las aplicaciones de Web se pueden desarrollar más rápida y eficientemente para facilitar el corto tiempo-al-mercado que es esencial en la Web.

5.3 Acceso a Objetos

En estos días, virtualmente todo nuevo desarrollo de aplicación se hace utilizando técnicas de modelado de objetos. Modelar datos como objetos permite a los desarrolladores pensar en los datos, de una manera natural e intuitiva. Y debido a que los objetos son modulares, con interfaces bien definidas, éstos son reutilizables y se pueden compartir entre aplicaciones, dando como resultado significativas ganancias de productividad.
Caché soporta un rango completo de técnicas de modelado de objetos, incluyendo herencia múltiple, encapsulación, polimorfismo, referencias, colecciones, parentesco y BLOB. Los Objetos de Caché se pueden crear con el Arquitecto de Objetos de Caché (una interface gráfica fácil de usar) ó a través del link bidireccional de Caché para Rational Rose (una popular herramienta de modelado de objetos). A diferencia de algunos sistemas de bases de datos para “objetos interdependientes", Caché permite un esquema de datos evolutivo. De esta manera las definiciones de objetos se pueden alterar para adaptarse a las necesidades cambiantes de sus aplicaciones. Y gracias a la Arquitectura de Datos Unificada de Caché, todos los Objetos Caché son compatibles automáticamente con ODBC.

Los Objetos de Caché también son compatibles con un amplio rango de herramientas y tecnologías orientadas a objetos. Pueden ser utilizados por desarrolladores en Java y C++, y por herramientas (tales como Visual Basic y Delphi) que usan la interface COM. Caché viene también con una interface CORBA bidireccional.

5.4. Acceso SQL

En su apogeo, las bases de datos interdependientes existían en todas partes, e incluso hoy en día, representan una mayoría de las bases de datos todavía en uso. Gran parte del software de aplicaciones, particularmente aquellos para el informe de datos y análisis, utilizan SQL como su lenguaje de consulta, y necesitaban una base de datos ODBC- o JDBC-flexible al otro extremo. A través de su acceso de datos SQL, Caché está disponible para todas estas aplicaciones. Además, la Entrada SQL de Caché permite a las aplicaciones Caché acceder a datos almacenados en bases de datos interdependientes – muy útil cuando hay necesidad de integrar datos desde una variedad de fuentes.

Algunos desarrolladores pueden desear trasladar aplicaciones desde una base de datos interdependiente hacia Caché para sacar ventaja del más alto nivel de gestión de Caché y su avanzada tecnología de objetos. Caché puede crear estructuras de datos desde una tabla de definiciones interdependientes contenida en archivos DDL. En virtud de la Arquitectura Unificada de Datos de Caché, cada tabla de definición se convierte en un objeto simple que se puede usar como éste, o como bloques de construcción para estructuras más complejas. Entonces, utilizando la Entrada SQL, los datos pueden ser transferidos desde las antiguas bases de datos interdependientes hacia Caché.

6. CONCLUSIONES

El primer paso en la realización del proyecto fue el diseño de la base de datos que modelase un esquema relacional y en el cual se pudiese almacenar toda la semántica posible asociada a los datos. Éste paso fue bastante costoso, ya que tras identificar las tablas que compondrían la BD, la tarea de identificar las relaciones era bastante compleja, puesto que las relaciones entre todas las tablas eran muy abundantes y si se mantenían todas las identificadas inicialmente se podría llegar a errores en la recuperación de los datos, por falta de consistencia en las relaciones.

Para optimizar las relaciones se realizó el modelo E/R de la BD y así se pudo identificar de una manera más clara, cuáles eran las relaciones indispensables, y cuáles de ellas eran redundantes y por lo tanto había que eliminarlas.

Finalmente se llegó a un modelo aprobado tanto por alumno como por profesor, aunque éste fue sufriendo algunos cambios a medida que se iba avanzando en el desarrollo de la aplicación.

El siguiente paso fue la elección del lenguaje de programación. Se decidió realizar la aplicación en Caché e implementar la Base de Datos en Access.
El lenguaje Caché era desconocido para mí y la labor de comenzar a programar con él resultó bastante costosa, puesto que nunca había manejado ningún lenguaje de programación orientado a eventos ni orientado a objetos.

Finalmente con ayuda de libros y manuales fui profundizando en los conocimientos, que finalmente me permitieron desarrollar la aplicación.

La primera fase del desarrollo de la aplicación fue el diseño de los formularios, a través de los cuales se permitiría insertar datos en la base de datos que contiene el modelo relacional, consultarlos e incluso modificar sus valores.

Esta labor resultó bastante compleja, especialmente en la parte del diseño del formulario de c*heck* y *disparadores* puesto que las posibilidades que ofrecen estos son muy amplias y hubo que limitarlas en cierta medida.

Por ejemplo cada vez que se comprueba el valor de un campo, éste puede ser comparado con el valor de otros campos o incluso con una lista de valores o con una subconsulta.

En el caso de los Check se ha intentado mantener todas estas posibilidades, pero por ejemplo se han limitado en la cláusula where de las acciones que puede llevar acabo un disparador. Se permite que el valor de un atributo se compare con un atributo de otra tabla o con un valor, y no con otra subconsulta, ya que ésta relación complicaría bastante el modelo de la BD.

Tras consultar dudas de este tipo al tutor se tomaron decisiones que permitieron obtener un modelo de datos y un diseño de formularios menos complejo.

El siguiente paso fue la programación de todas las funcionalidades en la aplicación, que al principio fue complicada puesto que el manejo de Recordset y objetos de conexión a bases de datos era desconocido para mí.

Se logrado cumplir la especificación de requisitos, a pesar de tener que limitar ciertas funcionalidades, debido a su grado de complejidad.

Finalmente, a nivel personal, éste proyecto me ha permitido conocer un nuevo lenguaje de programación, afianzar mis conocimientos de diseño de bases de datos y de SLQ y profundizar en el conocimiento de las bases de datos activas. Además trabajar con mi tutor de proyecto, que siempre estaba dispuesto a aclararme cualquier tipo de duda, ha contribuido a que esto sea una experiencia muy positiva.

7. DESARROLLOS POSTERIORES

A continuación se explican una serie de aspectos que podrían mejorar o ampliar las funcionalidades de la aplicación.

Una primera mejora sería implementar la BD en Oracle, que es un Sistema Gestor de Base da Datos que ofrece muchas más posibilidades que Caché.

La aplicación sólo permite insertar el modelo y la semántica de una base de datos en la base de datos a la que se conecta al iniciarse, por lo que para cada modelo de base de datos, del que se quiera generar el lenguaje SQL correspondiente es necesario disponer de una base de datos diferente. Esto implica tener que realizar varias copias de la BD sin datos, para poder ser utilizadas sin tener que volver a crear de nuevo todas las tablas y las relaciones que contiene.

Se podría mejorar este aspecto añadiendo una nueva tabla a la BD que almacene el nombre de la BD a la que se conecta la aplicación al inicio y que además que tenga una relación con las tablas que ésta contiene.

De esta manera, en una única BD se podrían almacenar los modelos y la semántica de varias bases de datos. Al iniciarse la aplicación y se seleccionaría la BD y el nombre de la BD con la que se va a trabajar y partir de ahí sólo se tendría acceso y se podrían dar da alta datos si éstos perteneciesen a la BD de la que se indicó el nombre al inicio.

Se podría añadir la funcionalidad de poder utilizar todas las funciones de las que dispone el lenguaje SQL, tales como LENGTH, SUBSTR, TO_CHAR, etc, para poder ampliar las posibilidades de semántica asociada a los datos.

8. BIBLIOGRAFÍA

A continuación se muestra el listado de libros y páginas Web que se han utilizado para el desarrollo de este proyecto:

Libros

- Tecnología y diseño de bases de datos. Piattini Velthuis, Mario G. Editorial Ra-Ma, año 2006.
- Fundamentos de bases de datos. Silberschatz, Abraham. Editorial McGraw-Hill, Año 2006.
- Active database systems: triggers and rules for advanced database processing. Widom, Jennifer. Morgan Kaufmann, Año 1996.

Webs

http://www3.uji.es/~mmarques/e16/teoria/cap1.pdf

http://www.bd.cesma.usb.ve/ci5311/sd05/apuntesParadigmasB.pdf

http://www.alu.ua.es/j/jmr36/Conectate/Base%20Datos/Apuntes2006.pdf

http://www.intersystems.es/es/prowebTemplates/baseTemplate.csp?pageID=20&mode=preview&version=current